그림으로 공부하는 오라클 구조

개정판

絵で見てわかるOracleの仕組み新装版

ⓒ 2019 Keiji Oda, Atsushi Sugita, Yumiko Yamamoto, Yuka Tsujii, Ryo Teramura

Original Japanese edition published by SHOEISHA Co.,Ltd.
Korean tanslation rights arranged with SHOEISHA Co.,Ltd. in care of The English Agency (Japan) Ltd. through Danny Hong Agency.
Korean translation copyright ⓒ 2020 by J-Pub Co., Ltd..

이 책의 한국어판 저작권은 대니홍 에이전시를 통한 저작권사와의 독점 계약으로 제이펍에 있습니다.
저작권법에 의해 한국 내에서 보호를 받는 저작물이므로 무단전재와 복제를 금합니다.

1쇄 발행 2020년 3월 31일

지은이 스기타 아츠시, 야마모토 유미코, 오다 케이지, 츠지이 유카, 테라무라 료
감수자 오다 케이지
옮긴이 이민재
펴낸이 장성두
펴낸곳 주식회사 제이펍

출판신고 2009년 11월 10일 제406-2009-000087호
주소 경기도 파주시 회동길 159 3층 3-B호
전화 070-8201-9010 / **팩스** 02-6280-0405
홈페이지 www.jpub.kr / **원고투고** jeipub@gmail.com
독자문의 readers.jpub@gmail.com / **교재문의** jeipubmarketer@gmail.com

편집팀 이종무, 이민숙, 최병찬, 이주원 / **소통·기획팀** 민지환, 송찬수 / **회계팀** 김유미
진행 및 교정·교열 이주원 / **내지디자인** 북아이 / **표지디자인** 미디어픽스
용지 에스에이치페이퍼 / **인쇄** 해외정판사 / **제본** 광우제책사

ISBN 979-11-88621-99-6 (93000)
값 25,000원

※ 이 책은 저작권법에 따라 보호를 받는 저작물이므로 무단전재와 무단복제를 금지하며,
이 책 내용의 전부 또는 일부를 이용하려면 반드시 저작권자와 제이펍의 서면 동의를 받아야 합니다.
※ 잘못된 책은 구입하신 서점에서 바꾸어 드립니다.

제이펍은 독자 여러분의 책에 관한 아이디어와 원고 투고를 기다리고 있습니다. 책으로 펴내고자 하는 아이디어나 원고가 있는 분께서는 책에 대한 간단한 개요와 차례, 구성과 저(역)자 약력 등을 메일로 보내 주세요. (보내실 곳: jeipub@gmail.com)

그림으로 공부하는 오라클 구조

개정판

스기타 아츠시, 야마모토 유미코, 오다 케이지, 츠지이 유카, 테라무라 료 지음
오다 케이지 감수 / **이민재** 옮김

Jpub 제이펍

※ 드리는 말씀

- 이 책에 기재된 내용을 기반으로 한 운용 결과에 대해 저자, 역자, 소프트웨어 개발자 및 제공자, 제이펍 출판사는 일체의 책임을 지지 않으므로 양해 바랍니다.
- 이 책에 등장하는 각 회사명, 제품명은 일반적으로 각 회사의 등록 상표 또는 상표입니다. 본문 중에는 ™, ©, ® 마크 등이 표시되어 있지 않습니다.
- 책 내용과 관련된 문의사항은 옮긴이나 출판사로 연락 주시기 바랍니다.
  - 옮긴이: mjlee@dataforum.io
  - 출판사: readers.jpub@gmail.com

차례

옮긴이 머리말 xiii
추천사 xv
시작하며 xvi
베타리더 후기 xix

CHAPTER 1 I/O와 디스크의 관계 1

1.1 오라클을 이해하기 위한 필수 키워드 2
1.2 오라클과 디스크(하드디스크) 3
1.3 디스크의 동작 4
1.3.1 어떻게 I/O의 대기 시간을 줄일까? 6
1.3.2 인덱스의 사용 예 7
1.3.3 랜덤 액세스 9
1.4 데이터를 보증하기 위한 디스크 12
1.5 요약 14

CHAPTER 2 오라클의 여러 프로세스 17

2.1 오라클의 역할 이미지 18
2.2 데이터베이스의 데이터는 모두의 것 20
2.2.1 실제 모습에 가까운 오라클의 이미지 20
2.2.2 엑셀과 DBMS의 차이 24
2.3 오라클이 여러 개의 프로세스로 구성된 이유 25
2.4 서버 프로세스와 백그라운드 프로세스의 역할 27
2.4.1 백그라운드 프로세스의 일 27
2.5 각 프로세스가 수행하는 처리 29
2.5.1 SQL문의 처리에 필요한 작업 30
2.6 요약 32

CHAPTER 3 캐시와 공유 메모리 37

3.1 어째서 캐시가 필요한 것인가? 38
3.2 그래서 캐시란 대체 무엇인가? 39
3.3 데이터는 블록 단위로 관리 41
3.4 캐시를 사용해서 인덱스 검색을 효율적으로 43
3.5 프로세스는 캐시를 공유 45
3.6 공유 메모리에 필요한 설정 48
3.7 공유 메모리는 어떤 식으로 보이는가? 50
3.8 버퍼 캐시를 정리하는 LRU 알고리즘 52
3.9 오라클뿐만이 아닌 OS나 스토리지에 대해서도 생각하자 54
3.9.1 OS의 버퍼 캐시와 가상 메모리의 차이 55
3.10 요약 58

CHAPTER 4 SQL문 분석과 공유 풀 61

4.1 SQL문의 분석과 공유 풀을 왜 배워야 하는가? 62
4.2 SQL문과 일반적인 프로그래밍 언어의 차이 62
4.3 서버 프로세스와 분석 63
4.3.1 비용을 계산하기 위한 기초 수치, '통계 정보' 65
4.4 실행 계획이 최적이라는 것을 판단하기 위해서는? 66
4.5 공유 풀의 동작과 구조 71
4.6 수치로 알아보는 분석과 공유 풀의 정보 74
4.7 요약 76

CHAPTER 5 오라클의 기동과 정지 79

5.1 기동과 정지를 왜 배워야 하는가? 80
5.2 오라클의 기동/정지의 개요 80
5.3 업무의 시작에 해당하는 오라클의 기동 81
5.4 인스턴스, 데이터베이스, 그리고 주요 파일의 구성 82
5.5 기동 처리의 흐름과 내부 동작 85
5.5.1 ① 기동 정지 상태에서 NOMOUNT로 전환 85
5.5.2 ② NOMOUNT로부터 MOUNT로 전환 86
5.5.3 ③ MOUNT에서 OPEN으로 전환 87
5.5.4 파일의 사용 순서를 확인해봄 88
5.5.5 기동 처리에서의 요점 90
5.6 업무 종료에 해당하는 오라클의 정지 91
5.7 데이터베이스를 수동으로 생성하기 93
5.8 요약 95

CHAPTER 6 커넥션과 서버 프로세스의 생성 97

6.1 애플리케이션에서의 접속을 왜 배워야 하는가? 98
6.2 오라클의 접속 동작 99
6.2.1 소켓의 동작 이미지 99
6.2.2 오라클에서 소켓의 동작 100
6.2.3 커넥션 처리 ①: 리스너를 기동 100
6.2.4 커넥션 처리 ②: 애플리케이션에서의 커넥션 102
6.2.5 커넥션 처리 ③: 서버 프로세스의 생성 104
6.3 접속 동작의 확인 106
6.3.1 tnsnames.ora 파일을 사용하지 않으면 어떻게 되는가? 106
6.3.2 데이터베이스 서버의 동작 108
6.4 정지나 리스너의 상태 확인 110
6.5 성능을 개선하기 위해서는? 111
6.6 요약 113

CHAPTER 7 오라클의 데이터 구조 115

7.1 오라클의 데이터 구조를 왜 배워야 하는가? 116
7.2 가변 길이 데이터를 관리할 프로그램을 만들기 위해서는? 117
7.2.1 필요한 데이터 구조 118
7.3 오라클의 데이터 구조 120
7.3.1 데이터 파일과 테이블의 관계 121
7.4 데이터 구조에는 어떤 것들이 있는가? 123
7.4.1 세그먼트 123
7.4.2 테이블스페이스 124
7.4.3 블록 안의 공간 125
7.4.4 ROWID 127
7.5 실제 흐름을 따라 각 동작을 확인 128
7.5.1 공간 할당하기 및 비어 있는 공간의 관리 128

7.6 프로세스에서 본 데이터 구조 130
7.7 요약 132

CHAPTER 8 오라클의 대기와 Lock 135

8.1 대기와 오라클의 Lock을 왜 배워야 하는가? 136
8.2 데이터베이스에 Lock이 필요한 이유 136
8.3 대기와 Lock 대기 139
8.3.1 Non-Idle 대기는 주의 141
8.3.2 Lock에 의한 대기는? 142
8.3.3 Deadlock의 구조 144
8.4 Latch의 구조 147
8.5 요약 150

CHAPTER 9 REDO와 UNDO의 동작 153

9.1 REDO와 UNDO를 왜 배워야 하는가? 154
9.1.1 A(Atomicity): 원자성 154
9.1.2 C(Consistency): 일관성 154
9.1.3 I(Isolation): 고립성 155
9.1.4 D(Durability): 지속성 155
9.2 지속성을 구현하기 위해서는 156
9.3 REDO와 UNDO의 개념 158
9.4 REDO의 구조 160
9.4.1 REDO의 요약 163
9.5 UNDO의 구조 163
9.6 여러 상황에서 REDO와 UNDO의 동작 165
9.6.1 롤백할 때의 동작 165
9.6.2 읽기 일관성에 동반되는 동작 165

9.6.3 커밋되지 않은 데이터를 읽어올 때의 동작 166
9.6.4 ORA-1555 에러가 발생했을 때의 동작 167
9.6.5 체크포인트의 동작 168
9.6.6 인스턴스 복구 시의 동작 169
9.7 요약 171

CHAPTER 10 백업/복구의 구조와 동작 175

10.1 백업/복구를 왜 배워야 하는가? 176
10.2 백업/복구에 필요한 지식의 복습 177
10.3 백업의 종류와 특징 179
10.3.1 온라인 백업의 순서 179
10.4 데이터베이스 손상의 예 181
10.4.1 디스크의 물리적인 고장으로 인한 데이터 파일의 손상 182
10.4.2 작업 실수 등으로 인한 OS상에서의 삭제나 덮어쓰기 182
10.4.3 어떤 문제로 인한 블록 손상 182
10.4.4 일시적인 데이터 파일의 접근 불가 182
10.4.5 하드웨어의 물리적인 고장이나 케이블의 문제 183
10.4.6 드라이버나 어댑터 카드 등의 제품 불안정에 의한 손상 183
10.5 기본적인 복구의 종류와 동작 183
10.5.1 인스턴스 복구와 미디어 복구 184
10.5.2 완전 복구와 불완전 복구의 차이 184
10.5.3 데이터베이스/테이블스페이스/데이터 파일/블록의 복구 184
10.5.4 복구할 필요가 없는 테이블스페이스도 존재한다? 186
10.6 기본적인 복구의 흐름(데이터베이스 전체의 복구) 188
10.6.1 ① 데이터베이스가 손상되었는지 확인 188
10.6.2 ② 재작업할 수 있도록 현재 상태 백업 189
10.6.3 ③ 필요한 데이터 파일과 아카이브 REDO 로그 파일 복원 189
10.6.4 복구 실행 191

10.7 그 외의 복구 192
10.7.1 불완전 복구란 무엇인가? 192
10.7.2 데이터베이스가 가동 중인 상태에서 테이블스페이스를 복구 193
10.7.3 컨트롤 파일 복구 195
10.8 요약 196

CHAPTER 11 백그라운드 프로세스의 동작과 역할 199

11.1 백그라운드 프로세스를 왜 배워야 하는가? 200
11.2 백그라운드 프로세스와 서버 프로세스의 관계 200
11.2.1 백그라운드 프로세스의 동작 200
11.2.2 슬립과 대기의 관계 203
11.3 DBWR의 동작과 역할 205
11.3.1 어떤 식으로 I/O를 하는 것인가? 206
11.3.2 DBWR의 수는 왜 장비마다 다른 것일까? 208
11.3.3 어떤 때 DBWR에 장애가 발생하는가? 208
11.4 LGWR의 동작과 역할 209
11.4.1 언제 I/O를 하는 거야? 209
11.4.2 어떤 때 LGWR에 장애가 발생하는가? 210
11.5 SMON의 동작과 역할 210
11.6 PMON의 동작과 역할 211
11.7 LREG의 동작과 역할 211
11.8 ARCH의 동작과 역할 212
11.9 그 외의 백그라운드 프로세스 214
11.9.1 CKPT 214
11.9.2 그 외의 여러 프로세스 214
11.10 요약 216

CHAPTER 12 오라클 아키텍처와 동작에 관한 Q&A 217

12.1 지금까지의 복습 218
12.2 오라클의 동작에 관한 질문 222
12.3 모니터링/운영에 관한 질문 223
12.4 해답과 해설 오라클의 동작에 관한 질문 224
12.5 해답과 해설 모니터링/운영에 관한 질문 233
12.6 요약 235

APPENDIX A 유스케이스로 배우는 오라클 239

A.1 A 씨에게 준비된 과제 240
A.2 오라클의 기동 240
A.3 리스너를 통한 접속 242
A.4 데이터 파일의 추가 245
A.5 백업하기 247
A.6 OS 명령어를 사용한 데이터 파일 삭제 253
A.7 현재 상태의 백업 255
A.8 복원 258
A.9 복구 260
A.10 데이터 파일의 제거 263
A.11 오라클의 정지 265

찾아보기 267

옮긴이 머리말

저는 이 책의 목차를 처음 읽었을 때 '내가 처음 오라클을 접했을 때 이런 도서가 있었다면 얼마나 좋았을까?'라는 생각이 들었습니다. 우리나라에 출간된 오라클 관련 도서들을 보면 고급 및 특급 수준의 독자만을 대상으로 하는 도서가 많습니다. 독자 중 상당수는 필독서로 추천받은 책들을 읽으며 '지금 내가 이런 수준의 책을 읽는 게 맞는 것일까?'라는 의문을 가진 적이 있을 겁니다. 저 역시 '내가 가진 기본 지식이 이 책을 읽기에는 부족하다는 것'을 느꼈고, 이는 마치 계단 몇 칸이 존재하지 않아 혼자서는 도저히 오르지 못해 목적지를 멍하니 바라만 보는 그런 느낌이 들었습니다.

하지만 저는 다행히도 그럴 때마다 많은 분의 도움을 받아 조금씩 계단을 올랐지만, 이러한 도움을 받지 못한 채 그 자리에서 헤매기만 한 분들도 많을 것으로 생각합니다. 그래서 저는 막 입문한 사람과 약간의 경험이 쌓여 초급을 벗어나고자 하는 사람들을 위한 도서를 번역하고 싶었습니다. 그 첫 번째 도서로, 초급 티를 벗기 위해 조금이라도 도움을 드리기 위한《나만 알고 싶은 오라클 실무 테크닉》1판을 2014년 9월에 번역, 출간하였습니다.

그리고 오라클을 접한 지 얼마 안 된 독자들에게 제대로 된 기본 지식과 공부의 방향을 알려주기 위해 개정판을 새로 번역하여 출간하게 되었습니다. 저자인 오다 케이지 님은 어려운 용어 대신에 오라클을 '창고 회사'에 비유해 초급 독자에게 조금이라도 더 쉽게 와 닿을 수 있게 집필하였습니다. 지금까지 오라클의 아키텍처를 제대로 이해하지 못하고 모호한 상태로 머릿속에 지니고 있었다면, 이 책을 읽음으로써 제대로 된 방향으로 나아가는 데 큰 도움이 될 것으로 생각합니다.

책과 관련된 번역의 오류, 제안뿐만 아니라 오라클과 관련된 궁금한 내용은 메일(mjlee@dataforum.io)을 통해 언제든지 연락주시기 바랍니다.

마지막으로 이번 책을 번역하는 데 처음부터 끝까지 많은 도움을 주신 (주)투씨에스지 윤병훈 팀장님께 감사의 말씀을 드립니다.

옮긴이 **이민재**

추천사

4차 산업혁명의 영향으로 인해 데이터의 관리와 활용의 중요성이 강조되는 현재 시점에서, 전 세계 기업용 데이터베이스 시장의 절반 이상을 점유하고 있는 오라클은 데이터베이스 분야에서 절대적인 우위를 선점하고 있습니다.

많은 오픈 소스나 상용 데이터베이스가 오라클 데이터베이스를 대체하려고 시도하고 있지만, 여전히 오라클 데이터베이스와의 기술적인 격차로 이를 따라잡지 못하고 있는 현실입니다.

그러다 보니 DBA나 DB 엔지니어가 되고자 오라클 데이터베이스를 통해서 이에 입문하려는 사람이 많고, 다른 데이터베이스를 사용하더라도 오라클 데이터베이스를 배우려는 사람들이 많습니다.

하지만 처음 시작하는 사람에게는 추상적인 오라클 데이터베이스의 아키텍처를 이해하는 것이 큰 장애물이기 때문에 오라클 데이터베이스는 입문하기 쉽지 않은 IT 기술(IT 솔루션) 중 하나입니다.

이 책은 창고 회사를 모델로 삼아서 오라클 데이터베이스의 아키텍처를 구체적으로 상상하기 쉽게 풀어서 설명하고 있으며, 오라클 데이터베이스를 접하며 막막함을 느꼈던 역자의 경험이 녹아 있는 번역으로 여러분의 오라클 데이터베이스 학습을 한결 수월하게 해줄 것입니다.

(주)투씨에스지 **윤병훈**

시작하며

안녕하세요? 일본 오라클에서 컨설턴트로 재직하고 있는 테라무라(寺村)라고 합니다. 저는 《그림으로 공부하는 오라클 구조》의 1판을 통해 데이터베이스의 구조를 배웠습니다. 그리고 이번에는 개정판의 집필에 참여하게 되었습니다.

이 책의 주제는 오라클 데이터베이스(이하, 오라클)의 아키텍처와 동작이며, 대상이 되는 독자는 다음과 같습니다.

- 오라클을 처음 배우는 분
- 오라클을 배웠지만, 아키텍처에는 자신이 없는 분
- 오라클을 배우다가 좌절한 분
- 오라클이 잘 이해되지 않아 고민 중인 분

이 책은 독자가 '기본적인 SQL문(SELECT/INSERT/UPDATE/DELETE/COMMIT/ROLLBACK)'을 알고 있다는 것을 전제로 썼습니다. 좀 더 구체적으로 말씀드리자면, SELECT문에서 사용되는 테이블의 조인(join)이나 데이터가 변경되면 COMMIT으로 확정하고, ROLLBACK으로 취소한다는 것 등을 이해한다면 충분합니다.

이 책은 다음과 같은 세 가지의 특징을 가졌으며, 아키텍처의 동작을 이해하기 위해서 간단한 명령어를 사용해 설명합니다.

- 아키텍처와 동작을 지속적으로 설명하여 머릿속에서 이미지를 떠올릴 수 있도록 한다.
- 가능한 한 오라클 이외의 용어(IT 기본 용어)를 사용해서 설명한다.
- 명령어를 설명하는 것은 이 책의 주목적이 아니다.

저는 2014년에 일본 오라클에 입사한 후, 4년 정도의 기간 동안 컨설턴트로서 프로젝트를 지원해 왔습니다. 오라클을 메인으로 두고서 OS, 네트워크, 스토리지 등의 여러 계층을 배우고 컨설팅 서비스에 활용하고 있습니다.

최근에는 가상화 기술(장비)이나 인프라를 자사에서 관리하지 않고 위탁하여 관리하는 클라우드 같은 서비스도 유행하고 있습니다. IT 업계에서의 유행은 매우 빠르게 바뀌지만, 배운 기술이 금방 도태되어 사용할 수 없게 되지는 않습니다. 새로운 기술이 등장하더라도 기초가 되는 아키텍처는 변하지 않는 경우가 많기 때문입니다.

하지만 아키텍처를 이해하지 못하고 단순히 명령어만을 외워서 '명령어는 이렇게 치면 된다'든지, '잘은 모르겠지만 원래 이런 거야'라는 식의 생각을 가진 엔지니어라면 기술의 변화를 따라가기가 힘들다고 생각합니다.

이 책에서 설명하는 데이터베이스인 오라클 역시 그 대표적인 사례이며, 지금까지 많은 버전이 공개되고 그때마다 여러 새로운 기능이 추가됐습니다.

최근에는 데이터베이스도 클라우드 환경에서 사용할 수 있게 되긴 했지만, 오라클 아키텍처를 제대로 이해하고 있는 엔지니어라면 대응하는 데 큰 문제가 없습니다. 오라클이라고 하더라도 결국 그 실체는 디스크나 네트워크를 사용하는 OS상에서의 애플리케이션에 지나지 않기 때문입니다.

이런 점들을 잘 이해하고 독자가 제대로 된 실력을 갖춘 오라클 엔지니어가 되기 위한 첫걸음을 내디딜 수 있도록 돕는 것이 이 책의 목적입니다. 더욱이 스스로 조사하고 배워 나가는 습관을 지니면 좋겠다고 생각합니다. 이 책의 구체적인 목표는 '**오라클의 기본적인 아키텍처를 이해하는 것**'입니다.

물론, 능력 있는 관리자가 되기 위해서는 아키텍처만으로는 충분하지 않습니다. 실제 사용하는 명령어와 같은 것들은 다른 서적이나 자료들을 활용하여 습득하는 게 좋습니다.

그리고 제 경험상 데이터베이스를 직접 만들어 보고 손상도 시켜보는 등의 여러 연습 과정은 배움에 있어 매우 중요합니다. 데이터베이스를 업무에서 사용하는 테스트 장비나 PC를 이용해서 구축하다 보면 명령어들도 몸에 자연스럽게 밸 것입니다.

단, 이런 연습을 할 때도 이 책에서 설명하는 아키텍처나 동작을 제대로 이해하고

있다면 그 체험의 깊이가 달라질 것이므로 꼭 아키텍처나 동작을 머릿속에 그려가면서 데이터베이스를 구축하세요.

모쪼록 오라클 아키텍처의 세계를 그림으로 보고 즐기면서 체험하여, 독자분들의 실력에 도움이 된다면 행복할 것 같습니다.

저자들을 대표하여 **테라무라 료**

베타리더 후기

※ 개정판과 1판의 기본 형태는 거의 유사하여 1판을 베타리딩하신 분들의 후기를 개정판에도 그대로 실었음을 밝힙니다.

강미희(휴인시스텍)

그림과 함께 설명되어 있어 기초를 쉽고 확실하게 배울 수 있었던 책이었습니다. 팁과 칼럼을 통해 조금 더 깊은 내용을 알 수도 있었고, 각 장의 요약을 통해 한 장의 내용을 머릿속에 다시 상기할 수 있어서 좋았습니다. 그리고 마지막 장에서의 Q&A를 통해서는 어떻게 응용해야 할 것인가에 대해서도 어렴풋하게나마 정리할 수 있어서 많은 도움이 되었습니다.

고승광(플랜티넷)

과거 오라클을 책으로 처음 접했을 때 오라클 아키텍처가 OS처럼 생각되어서 바로 포기했던 기억이 납니다. 그런데 이 책은 오라클 아키텍처를 일상(회사/창고)에 비교해서 아주 알기 쉽게 설명하고 있습니다. 이 책을 읽고 나서 《나만 알고 싶은 오라클 실무 테크닉》을 읽어 보면 학습 효과가 더 클 듯합니다.

김주현(위즈베이스)

오라클의 내부 작동 원리도 이해하고 나면 전혀 어렵지 않습니다. 사람들이 오라클 아키텍처를 어려워하는 이유는 무턱대고 암기하려고 들기 때문입니다. 이 책은 '왜?'라는 질문을 바탕으로 구성되어 있어서 끝까지 읽고 나면 머릿속에 SQL문이 오라클 내부에서 처리되는 모습이 자연스럽게 그려질 겁니다. 또한, SQL의 처리가 늦어지는 원인과 해결책을 찾는 데 필요한 기본 지식도 얻을 수 있습니다.

손정호(한의사)

《그림으로 공부하는 오라클 구조》라는 제목에 어울리게 그림으로 된 설명이 적재적소에서 독자의 이해를 돕고 있습니다. 덕분에 평소 데이터베이스나 오라클에 대한 지식이 부족했는데도 막힘없이 재미있게 볼 수 있었습니다. 오라클의 내부 구조가 궁금하지만 너무 두껍고 어려운 책을 잡기는 엄두가 안 나시는 분들께 부담 없이 일독을 권하고 싶습니다.

이상현(SI개발자)

오라클 책을 접하니 사회 초년 시절에 프로그래밍과 함께 SQL의 매력에 빠져들었던 기억이 잠시 들었습니다. 이 책을 보기 전에는 '그림으로 오라클을 설명해 봐야 얼마나 깊이가 있겠어?'라고 생각했는데, 읽으면서 점점 창피하더군요. 데이터를 뽑는 데에만 신경을 쓰느라 미처 어떻게 구성되고 관리되는지를 전혀 생각하지 못했기 때문일 겁니다. 그간 모르고 있던 부분을 선배, 친구가 쉽게 하나하나 설명을 해준다는 생각이 들 정도로 쉽게 다가왔습니다.

이재빈(연세대학교 정보대학원)

글보다는 그림 등의 시각적 자료를 통해 정보를 얻는 데 능숙한 세대에게 적합한 콘텐츠로, 정확한 지식을 간결하게 전달하는 데 많은 노력을 기울인 책이라고 생각합니다. 오라클의 아키텍처를 모르는 사람이라도 이 책 한 권으로 전반적인 그림을 그릴 수 있을 것이라 확신이 들었습니다. 정말 쉽게 오라클의 아키텍처를 이해할 수 있어서 좋았습니다.

제이펍은 책에 대한 애정과 기술에 대한 열정이 뜨거운 베타리더들로 하여금
출간되는 모든 서적에 사전 검증을 시행하고 있습니다.

CHAPTER 1

I/O와 디스크의 관계

이 책은 기본적인 오라클 아키텍처를 그림과 함께 알기 쉽게 설명하였습니다. 1장에서는 오라클의 기본 기능을 이해하기 위한 '오라클 필수 키워드'를 선정하고 이것을 토대로 I/O(Input/Output, 입출력)와 디스크의 관계를 자세하게 설명합니다. 일반적으로 디스크와 I/O의 동작을 특별히 의식한 적은 없겠지만, 머릿속에서 동작을 그림으로 떠올릴 수 있다면 오라클에서 발생하는 여러 현상을 이해하기가 수월해집니다. 자, 그러면 응용력이 있는 오라클 엔지니어가 되기 위한 첫걸음을 내디뎌봅시다.

1.1 오라클을 이해하기 위한 필수 키워드

우선은 오라클을 이해하기 위한 세 가지 키워드를 소개하겠습니다. 이 키워드는 이번 장뿐만 아니라 이 책의 여러 장에서 반복적으로 나옵니다.

키워드

① 병렬 처리를 가능케 하고 높은 처리량을 실현한다.
② 응답 시간(response time)을 중시한다.
③ 커밋(COMMIT)한 데이터는 지킨다.

오라클을 포함한 모든 DBMS(DataBase Management System, 데이터베이스 관리 시스템)의 내부 구조는 매우 복잡합니다. DBMS가 복잡해지는 이유는 위에서 이야기한 세 가지 특성을 모두 충족시켜야 하기 때문입니다. 이 세 가지 특성은 상반된 성향이 있어서 동시에 모두 만족시키기가 매우 어렵습니다. 예를 들어, '커밋(COMMIT)한 데이터는 지킨다'를 만족하기 위해 커밋하는 순간 데이터를 디스크에 기록하고 싶지만, 그렇게 하면 응답 시간이 나빠지게 됩니다.

'병렬 처리를 가능케 하고 높은 처리량을 실현'하는 것도 간단하지 않습니다. 병렬 처리할 때는 모순되는 처리가 발생하지 않도록 Lock[주1]이 필요하며, 그로 인해 성능이 저하되는 경우도 발생합니다.

이 책에서는 장마다 주제를 정해서 오라클의 'DBMS로서의 기본 기능'을 설명합니다. 최근의 오라클은 기능이 더 풍부해졌지만, 이 기능들은 모두 기본 기능들을 토대로 하고 있습니다.

그리고 이 책에 나오는 수치들은 독자의 이해를 돕기 위해 사용한 참고 수치이며, 단순한 기준값 정도로만 생각하세요.

주1 데이터를 읽고 쓸 때 데이터의 정합성을 유지하기 위해 데이터의 읽기/쓰기를 일시적으로 제한하는 것

1.2 오라클과 디스크(하드디스크)

오라클은 DBMS이며, 오라클의 관점에서 데이터베이스는 오라클이 디스크에 저장하고 관리하는 데이터를 의미합니다. 오라클은 디스크에서 데이터를 읽어오고, 필요한 처리를 한 후 다시 디스크에 기록합니다. 즉, 오라클이 다루는 데이터는 디스크에서 꺼내오고 다시 디스크로 돌아갑니다. 그래서 오라클과 디스크는 떼려야 뗄 수 없는 관계입니다.

1장에서는 디스크를 설명합니다. 만약 고장 나도 상관없는 디스크를 가지고 있다면 분해한 후 내부 구조를 살펴보면서 읽어 주세요. 단, 한번 디스크를 분해하면 다시는 사용할 수 없으므로 주의합시다.

그러면 먼저 디스크가 어떻게 동작하는지를 머릿속에 이미지로 그려보기 위한 첫 번째 그림을 설명하겠습니다. 먼저 디스크를 레코드판이나 CD라고 생각하고 이미지를 떠올려주세요(그림 1.1).

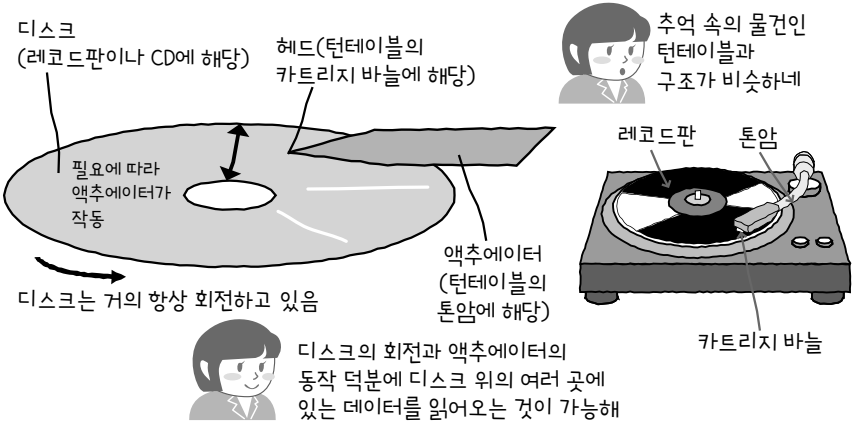

그림 1.1 **디스크와 레코드판의 그림**

디스크는 거의 항상 회전하고 있으며, 그 위로 헤드가 움직여서 데이터를 읽거나 기록합니다. 레코드판과 다른 것은 원반이 여러 개 겹쳐 있다는 것과 액추에이터가 양면을 사용한다는 것이지만, 여기서는 중요하지 않습니다. 또 하나의 차이점으로는 속도를 들 수 있습니다. 음악에 비유하자면, 트랙의 첫머리를 찾는 작업을 1초에 100번 할 수 있을 정도로 헤드가 빠르게 움직입니다. 그리고 1분에 1만 번 정도의 엄청난 속도로 회전합니다(그림 1.2).

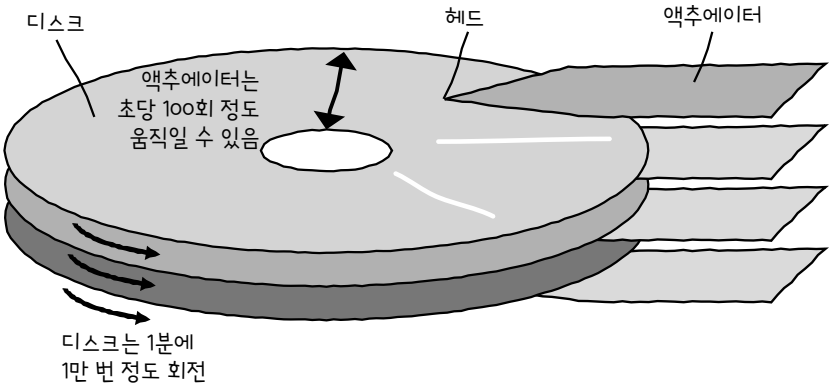

그림 1.2 **좀 더 현실에 가까운 디스크의 이미지**

1.3 디스크의 동작

무서울 정도로 빠르게 움직이는 레코드판의 이미지를 떠올릴 수 있게 되었으니, 다음으로는 I/O 처리에 필요한 디스크의 동작을 살펴보겠습니다. 데이터를 읽기(또는 기록하기) 위해서는 원하는 데이터가 저장되기 시작한 첫머리를 반드시 찾아야 합니다. 이렇게 원하는 위치를 찾는 작업을 디스크 용어로 '탐색(seek, 시크)'이라고 부릅니다.

그 후에 디스크에서 원하는 정보를 읽을 수 있는 위치가 회전해서 다가올 때까지 기

다리다가 해당 위치가 다가오면 데이터를 읽고 씁니다(그림 1.3). 이렇게 원하는 위치가 다가올 때까지 기다리는 시간을 '회전 대기 시간(rotational latency time)'이라고 부릅니다.

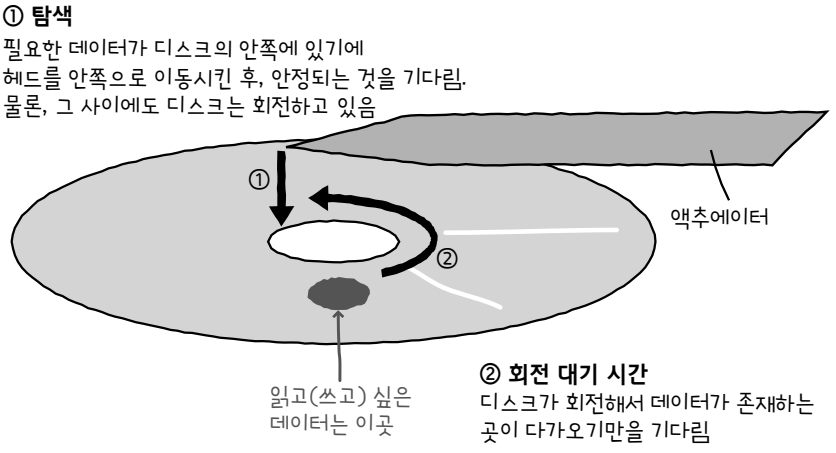

그림 1.3 **I/O 처리에 필요한 동작**

메모리에 접근할 때는 나노초(ns) 단위로 수행할 수 있으나, 디스크에 접근할 때는 밀리초(ms) 단위의 시간이 필요합니다. 사람의 입장에서 보면 디스크에 접근하는 밀리초라는 단위도 매우 빠른 것처럼 보이겠지만, 컴퓨터의 입장에서는 매우 느린 속도입니다.

이유를 간단하게 설명하자면 메모리는 전기 신호로 작업을 처리하지만, 디스크는 기계 동작이 필요하기 때문입니다. DBMS의 입장에서 보면 디스크 I/O는 반드시 필요한 것이면서도 처리 시간을 단축하기 위해서는 가능한 한 줄여야만 하는 부분이기도 합니다.

여기서는 탐색 시간과 회전 대기 시간을 합쳐 10밀리초가 걸린다고 가정하겠습니다. 또한, 디스크의 전송 속도(헤드를 움직이지 않고 계속 데이터를 읽고 쓸 수 있을 때의 처리량)는 20MB/초라고 가정하고 설명하겠습니다.

1.3.1 어떻게 I/O의 대기 시간을 줄일까?

그러면 어떤 방식으로 I/O의 대기 시간을 줄일 수 있을까요? 그 답을 알아보기 위해 먼저 시퀀셜 액세스에 관해 설명하겠습니다. 시퀀셜(sequential)은 '순서를 따라서'라는 의미의 '순차'를 뜻하며, 시작점에서부터 마지막까지 중간 부분을 빠트리지 않고 전부 액세스(읽기/쓰기)하는 것을 의미합니다. 메모리에 테이블의 데이터가 없으면 풀 스캔(Full Scan, 테이블의 모든 데이터를 읽어오는 것)할 때 시퀀셜 액세스가 발생합니다(그림 1.4). 이 두 가지 용어(시퀀셜, 풀 스캔)는 데이터베이스를 이해하기 위해서 매우 중요하므로 꼭 기억해두세요.

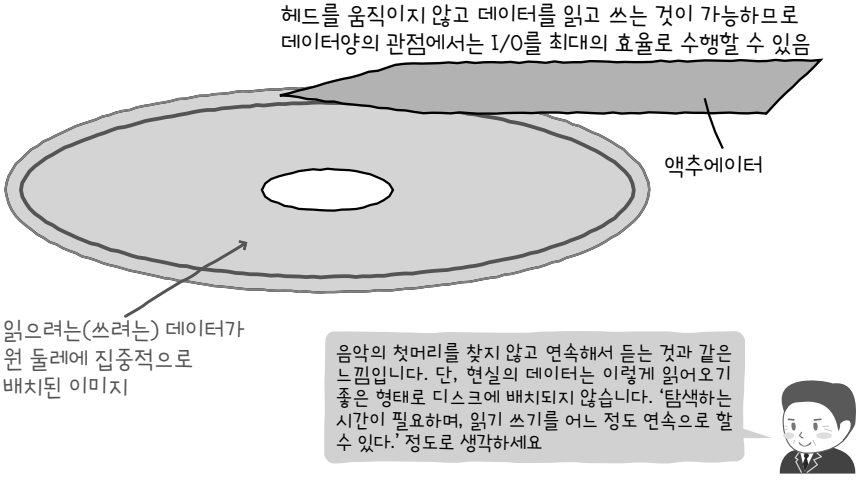

그림 1.4 **시퀀셜 액세스**

테이블의 크기를 1GB라고 가정하고 모든 데이터를 시퀀셜 액세스로 가져온다면 50초(크기/전송 속도 = 1,000MB/20MB/초)가 걸립니다. 테이블 크기가 100MB라고 하더라도 5초가 걸립니다. 데이터를 가져오는 시간이 이렇게 오래 걸리면 수행하는 대부분의 SQL 성능이 만족스럽지 못합니다.

그래서 인덱스(index, 색인)라는 발상이 나오게 되었습니다. 책에서 무엇인가를 조사하고 싶을 때 여러분은 어떻게 하는지요? 책의 페이지를 처음부터 끝까지 읽어 나가면

서 찾는 분은 거의 없을 테고, 대부분은 색인을 이용할 것입니다. 색인에는 키워드가 순서대로 나열되어 있으며, 해당 페이지 번호도 함께 기재되어 있습니다. 그 페이지 번호를 이용해서 원하는 페이지를 빠르게 찾은 후 필요한 정보를 얻을 수 있습니다.

데이터베이스의 인덱스도 마찬가지입니다. 데이터베이스의 인덱스에는 검색할 때 사용하는 키 값(SQL문의 WHERE절에 적는 조건의 값을 말함)과 그 키가 존재하고 있는 위치가 기록되어 있습니다(그림 1.5).

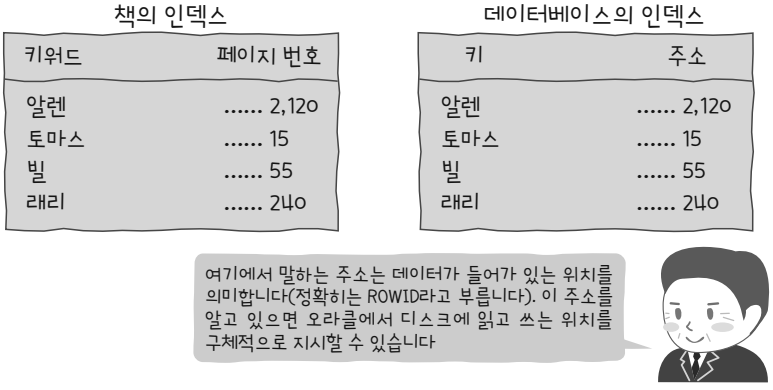

그림 1.5 **인덱스의 이미지**

1.3.2 인덱스의 사용 예

예를 들어, 래리에 관한 내용을 책에서 찾아보고 싶을 때는 인덱스에서 해당 내용이 240페이지에 쓰여 있다는 것을 확인한 후 해당 페이지를 펼쳐서 래리의 정보를 확인할 수 있습니다. SQL문도 마찬가지로, WHERE절에 '래리'의 정보를 찾고 싶다는 것을 작성하고 SELECT절 뒤에 알고 싶은 항목(예를 들면 '소속 회사')을 적습니다.

```
SELECT "소속 회사" FROM "개인 데이터" WHERE "이름" = '래리';
```

인덱스를 사용해서 이 SQL문을 처리할 때는 우선 '이름'이 기록되어 있는 인덱스를 조사합니다. 그 결과로 주소(ROWID)를 알아낼 수 있고, 그 주소를 토대로 데이터를 읽

어옵니다. 읽어온 데이터 안에는 래리의 데이터가 전부 모여 있으므로 그 안에서 '소속 회사'의 데이터를 사용자에게 반환합니다(그림 1.6).

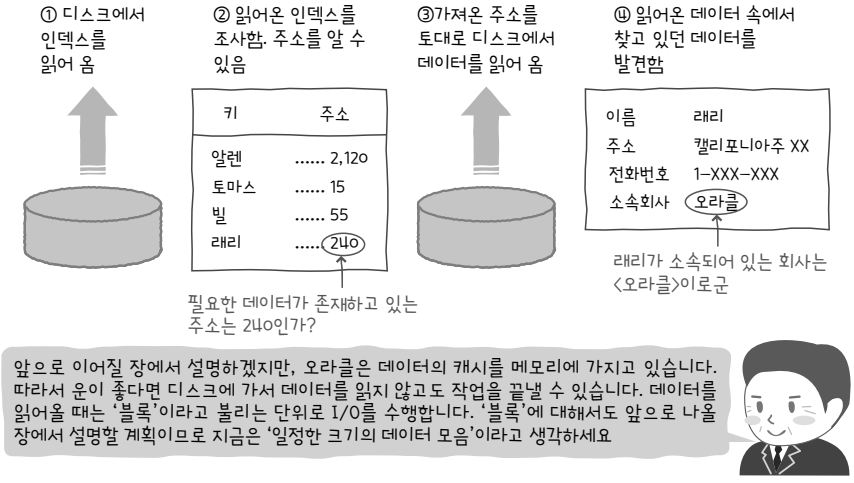

그림 1.6 인덱스를 이용한 SQL 처리의 흐름

앞서 설명한 대로 인덱스는 매우 편리한 기능입니다. 그렇다면 인덱스 자체의 크기가 커지면 크기가 큰 테이블을 조회하는 것과 마찬가지로 작업에 필요한 시간이 늘어나게 될까요? 답을 먼저 말씀드리자면, 그런 일은 발생하지 않습니다. 왜냐하면, 오라클의 인덱스는 '인덱스에 인덱스를 추가하는 것'처럼 여러 계층으로 구성되기 때문입니다(그림 1.7). 이런 부분이 책에서 사용하는 색인과 다른 점입니다.

이렇게 여러 계층으로 구성된 구조를 트리(tree) 구조라 부릅니다(나무를 거꾸로 한 모양처럼 보일 것입니다). 트리 구조의 장점은 인덱스의 필요 없는 부분을 읽지 않고 검색(작업)을 끝낼 수 있다는 것입니다.

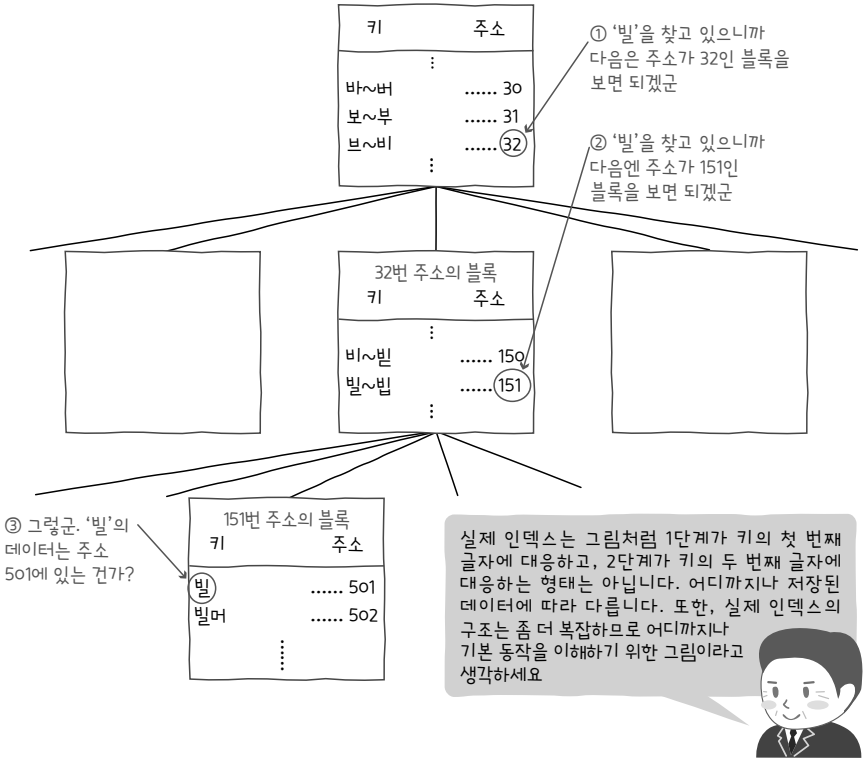

그림 1.7 **여러 계층을 가진 인덱스**

1.3.3 랜덤 액세스

다시 I/O 이야기로 돌아오면, 인덱스를 사용할 때는 필요한 부분만 읽어오면 충분하지만, 필요한 부분이 디스크 위에 연속적으로 존재하는 경우는 거의 없습니다. 따라서 헤드를 움직여가면서 띄엄띄엄 접근하게 됩니다. 이렇게 접근하는 방식을 '랜덤 액세스(random access)'라고 하며, 시퀀셜 액세스와 반대의 의미가 있습니다.

디스크의 관점에서 생각해보면 랜덤 액세스는 비효율적인 부분이 존재합니다. 탐색하는 작업과 회전 대기로 인해 데이터에 띄엄띄엄 접근할 때마다 어쩔 수 없이 시간이 소모되기 때문입니다. 가령 오라클의 블록 크기를 8KB, 1초에 탐색할 수 있는 횟수를

100회라고 가정한다면, 1초 동안 겨우 800KB밖에 읽어올 수 없습니다. 이것은 디스크의 전송 속도(헤드를 움직이지 않고 읽고 쓸 수 있을 때나 가능한 처리량)를 20MB/초라고 가정했을 때, 데이터를 탐색하지 않고 한 번에 가져올 수 있는 20MB라는 크기에 비하면 25분의 1에 불과합니다.

이것을 음악 청취에 비유한다면, 첫머리를 찾아가는 것을 빈번하게 반복하다 보니 한 시간 중 3분 정도밖에 음악을 듣지 못한 것입니다. 남은 57분은 첫머리를 찾아다니느라 낭비한 셈입니다.

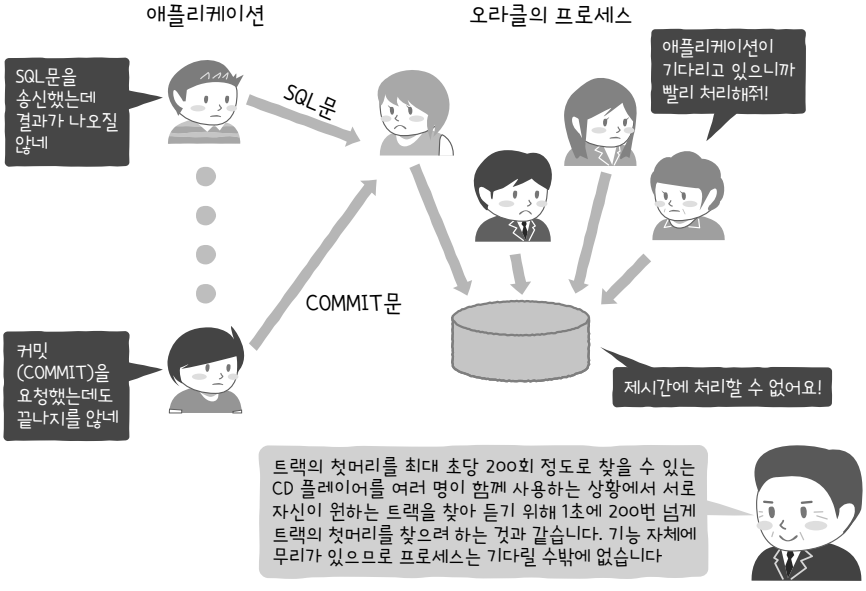

그림 1.8 **디스크의 수가 적으면 디스크 자체가 병목이 됨**

실제로 데이터 전송의 효율 측면에서 바라보면 DBMS의 I/O도 마찬가지입니다. 탐색을 반복하기 때문에 DBMS(특히 OLTP[주2] 시스템)에서 사용하는 디스크는 IOPS(I/O Per Sec, 초당 수행 가능한 I/O의 횟수) 지표가 중요합니다. 그리고 대부분의 디스크는 IOPS가 100회나 200회 정도이기 때문에 한 개나 두 개의 디스크만을 사용해서 데이터베이스를 구성하면 부하가 집중적으로 발생했을 때 요청이 오는 것을 탐색이 따라잡을 수 없으므로 디스크에 병목 현상이 발생합니다(그림 1.8).

팁

'인덱스를 사용하는 것이 유리한 경우는 어째서 필요한 데이터가 전체 데이터의 15% 미만'이라고 하는 걸까?

그 이유는 시퀀셜 액세스와 랜덤 액세스의 특성에 있습니다. 테이블의 데이터가 대량이고 그중 한 개의 로우(행)를 꺼내야 한다면 당연히 인덱스를 통한 액세스가 빠릅니다.

그에 비해 모든 데이터를 보려고 할 때 매번 인덱스를 찾은 후 데이터를 찾아가면 오히려 속도가 느려집니다(색인을 일일이 찾아가면서 책의 내용 전체를 읽는 독자분은 안 계실 겁니다).

만약 데이터가 50%라면 어떨까요? 여기에서 중요한 점은 '디스크에서의 랜덤 액세스는 데이터를 읽어오는 효율성이 시퀀셜 액세스보다 떨어진다'라는 특성입니다. 예를 들어, 테이블에 2만 건인 데이터가 저장되어 있다고 하고 그중 절반인 1만 건을 꺼낸다고 가정하겠습니다. 여기서 한 로우는 8KB입니다. 이때 지금까지 사용해 왔던 디스크의 성능을 기준으로 계산해보면, 랜덤 액세스로 약 100초가 걸립니다(인덱스는 빈번하게 사용되고 있으므로 캐시(cache)에 보관되어 있다고 가정합니다). 그것과 비교하면 2만 건 전부를 읽어오는 시퀀셜 액세스로는 모든 데이터(2만 건)를 디스크에서 읽어온다고 해도 약 8초면 끝납니다. 즉, 디스크 특성상 모든 데이터가 아니더라도 일정 크기 이상의 데이터를 읽는다면 시퀀셜 액세스를 사용하여 테이블을 풀 스캔하는 편이 빠릅니다.

단, 실제로는 캐시에 데이터가 보관된 경우도 있으며, 한 개의 블록에 여러 로우의 데이터가 보관되어 있어서 1회의 I/O로 많은 데이터를 읽을 수 있는 경우도 있습니다. 더욱이 인덱스의 데이터를 디스크에서 읽어 와야 하는 경우도 있으므로 '15%가 임계치'라고 단순하게 말할 수는 없습니다. 어디까지나 대략적인 기준 정도로만 생각하세요.

주2 OLTP(OnLine Transaction Processing). 철도의 좌석 예약 시스템이나 은행의 입출금 처리라고 생각하면 됩니다. 많은 단말에서 온라인(네트워크를 통해서)으로 크지 않은 데이터를 읽거나 쓰는 작업을 하고 결과를 즉시 회신해야 하는 시스템 형태를 말합니다. 데이터가 크지 않다는 것과 '즉시'라는 부분에서 알 수 있듯이, 인덱스를 사용해야 하는 시스템 형태입니다.

1.4 데이터를 보증하기 위한 디스크

데이터는 오라클의 프로세스가 비정상적으로 종료되었어도 무사해야 합니다. 이 점이 DBMS와 다른 프로그램과의 차이점 중 하나입니다. 예를 들어, 엑셀(Excel)에서는 저장한 시점 이후의 데이터는 없어집니다. 이미 알고 계시는 바와 같이 데이터는 프로그램이 비정상적으로 종료되거나 전원 버튼을 눌러서 전원이 끊어져도 사라집니다.

그것과 비교하면 DBMS는 어떤 장애가 발생해도 견뎌내야만 합니다. 데이터는 CPU나 메모리가 고장이 나거나 정전이 발생했다 하더라도 손실되어서는 안 됩니다. 또한, 컴퓨터의 주기억장치 내의 정보는 전기이므로 정전이 발생하면 메모리에서 사라집니다. 이런 가혹한 조건 속에 키워드에서 소개한 '커밋(COMMIT)한 데이터는 지킨다'라는 특성을 어떻게 구현하고 있는 것일까요?

답은 간단합니다. 오라클은 데이터를 변경한 후에 COMMIT이라고 입력하면 데이터를 디스크에 기록합니다(그림 1.9). '그러면 느리지 않을까?'라고 생각할 수도 있지만, 다행히 속도를 빠르게 하기 위한 장치가 마련되어 있습니다. 오라클을 필두로 하는 DBMS들은 성능 향상을 위한 이런 장치들이 존재해서 그 구조가 복잡하다고 볼 수 있습니다. 이 부분에 관해서는 뒤에 나올 장에서 설명하겠습니다.

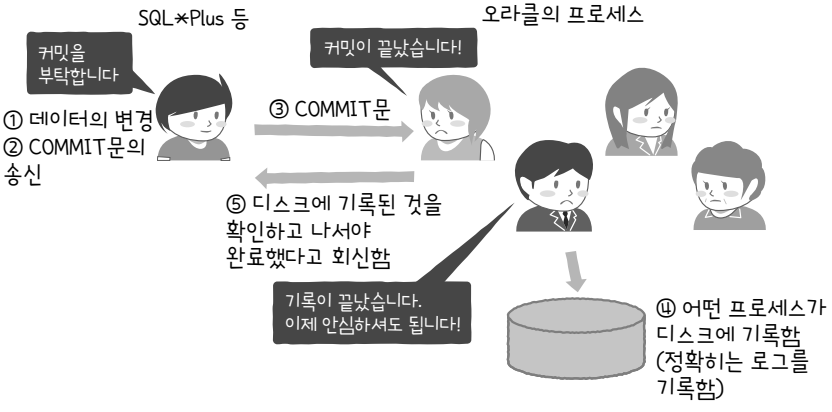

그림 1.9 **커밋을 수신하면 데이터를 디스크에 기록한다**

칼럼 '시퀀셜'이란 어떤 의미인가?

오라클은 여러 정보를 대량으로 보관하고 있습니다. 그중에는 I/O를 얼마만큼 수행했는지에 대한 정보도 기록되어 있지만, 내용을 상세하게 살펴보면 헷갈리기 쉬운 부분들이 있습니다.

예를 들어, 시퀀셜 액세스는 'db file scattered read'라고 표시되며, 랜덤 액세스는 'db file sequential read'라고 표시됩니다. 'scattered'는 '분산됐다'라는 의미를 가지고 있으며, 'sequential'은 '순차적인, 연속적인'이라는 의미를 가지고 있습니다. 액세스하는 방식을 생각해보면 반대로 되어 있는 것이 아닌가라는 생각도 들지만, 표시가 잘못되어 있는 것은 아니며 다음과 같은 의미가 있습니다.

오라클은 블록 단위로 데이터를 읽고 쓰며, 메모리에 배치해둡니다.

시퀀셜 액세스는 '순차로(순서대로) 읽는다'라는 의미로, 여러 블록을 한꺼번에 빠짐없이 읽어옵니다. 이때 읽어온 여러 블록은 메모리상에 연속되지 않은(분산된) 형태로 놓입니다. 따라서 'scattered'라고 표현합니다. 그에 반해 랜덤 액세스에서 읽어오는 데이터 블록은 한 번에 한 개이며, 메모리 공간에 반드시 연속적으로 놓입니다. 따라서 'sequential'이라고 표현합니다. 한번 읽어온 블록을 메모리에 어떻게 배치하는지에 따라 표현하는 방법을 다르게 한 것입니다.

저는 시퀀셜이라는 이름이 '시퀀셜 I/O'만을 의미하기를 원하지만, 이미 정해진 용어이기 때문에 변경할 수는 없을 것입니다. 혼동하기 쉬운 부분이므로 여러분도 잘 기억해두시기 바랍니다.

참고 《Database Performance Tuning Guide》의 10.3.3 db file scattered read와 10.3.4 db file sequential read

1.5 요약

이번 장에서 여러분들이 잘 이해하고 넘어가야 하는 부분은 다음의 세 가지입니다. 독자 여러분의 머릿속에 이 이미지들이 잘 떠오를 수 있었으면 합니다.

- 디스크가 회전하고 있는 이미지
- 헤드가 움직이는 것과 회전해 오기를 기다리고 있어서 I/O에 시간이 걸리는 이미지
- 인덱스로 인해 해당 데이터에 빠르게 접근할 수 있다(테이블을 맨 앞에서부터 끝까지 풀 스캔하지 않아도 된다)는 이미지

이번 장에서는 디스크가 느리다는 것, 그리고 인덱스로 인해 높은 효율로 데이터에 접근할 수 있다는 것을 알게 되었습니다. 이런 내용이 어떻게 전개되는지는 이후의 장에서 설명할 것입니다. 한 개만 먼저 소개해두자면, 버퍼 캐시(buffer cache)는 SQL문이 디스크의 성능에 영향을 가능한 한 받지 않게 하기 위해 존재합니다. 해당 내용은 3장에서 자세하게 설명하겠습니다.

용어

24365, 24/7, 온-프레미스

현장에서는 문서로는 접할 수 없는 용어가 사용되기도 합니다. 그런 용어들을 주변 사람들에게 물어보기란 쉽지가 않습니다. 그러므로 여기서는 현장에서 사용되는 몇 가지 용어를 소개하겠습니다.

- **24365, 24/7**

24시간 365일 가동되는 시스템을 의미합니다. 고가용성과 보안을 유지하기 위한 대책이 중요합니다.

- **온-프레미스(On-premise)**

장비들을 직접 보유하고 IDC 등에서 운영하는 형태를 의미합니다. 최근에는 인터넷에 가상 서버를 두고 사용하는 '클라우드' 환경도 유행하고 있는데, 이를 구분하기 위한 용어라고 생각하세요.

칼럼

저장 장치(스토리지)의 트렌드

데이터를 보관해두는 저장 장치로는 HDD(Hard Disk Drive)가 가장 오래 사용되어 왔습니다. 최근에는 이 HDD를 대신해서 빠르게 보급되고 있는 것이 SSD(Solid State Drive)입니다.

SSD는 플래시 메모리를 사용한 저장 장치입니다. HDD의 자기 디스크와는 다르게 주소로 표시되는 저장 공간에 데이터를 보존합니다. 따라서 디스크의 회전 등 물리적인 동작을 필요로 하지 않고, 빠르게 데이터를 보존하고 꺼내올 수가 있습니다. 또한, 디스크에서 필요한 회전 등의 기계적인 구조 역시 필요 없으므로 충격에도 강합니다. 그렇지만 SSD는 용량 대비 가격이 고가입니다.

최근에는 용량이 많이 필요한 시스템에서는 HDD를 사용하고, 성능을 중시하는 시스템에서는 SSD를 선택하는 경우가 많아지고 있습니다. 앞으로 기술이 진보함에 따라 점차 SSD의 가격이 낮아지고 용량도 커질 것을 기대합니다.

CHAPTER 2

오라클의 여러 프로세스

1장에서는 디스크의 동작이 느리다는 것과 인덱스를 사용하면 높은 효율로 데이터에 접근할 수 있다는 것을 설명했습니다. 이번 장에서는 오라클을 '창고 업자'라고 가정하고 SQL*Plus 등의 각종 프로그램과 데이터를 주고받는 구조를 살펴보도록 하겠습니다. 이 장을 통해 오라클에는 어떤 프로세스가 있고, 어떤 식으로 병렬 처리를 수행하는지를 이해하시면 됩니다.

2.1 오라클의 역할 이미지

초급 독자들의 이해를 돕기 위해 오라클이 어떤 역할을 하는지를 먼저 설명하겠습니다. 여기서는 오라클을 '창고 업자'라고 가정하겠습니다. 창고 업자란 고객의 짐을 받아서 맡아주고, 짐을 창고에 보관하고, 고객의 요청에 따라 짐을 고객에게 돌려주는 업무를 하는 회사입니다(그림 2.1).

오라클을 포함한 DBMS(데이터베이스 관리 시스템)도 기본적인 동작은 '데이터를 짐처럼 맡아서 보관하고, 요구에 따라 데이터를 반환한다'라는 점에서 꽤 비슷합니다.

DBMS를 사용한 애플리케이션 개발 경험이 없는 독자들은 이어지는 칼럼 'DBMS를 사용한 애플리케이션에 관해서'를 참고해주세요.

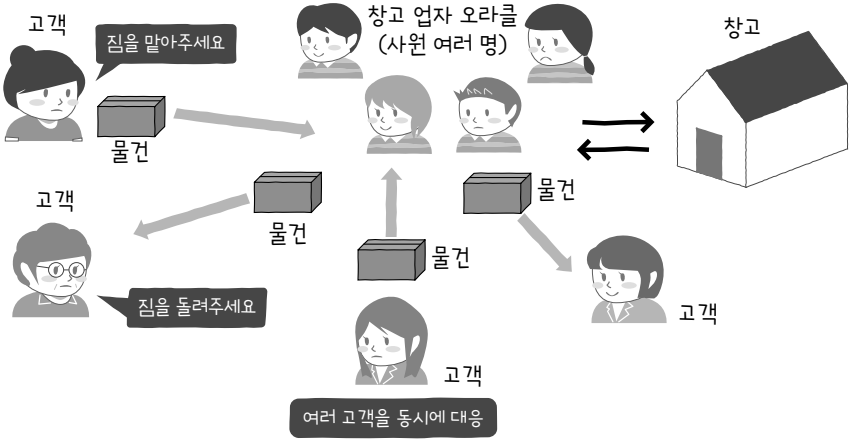

그림 2.1 **'창고 업자'로 표현해본 오라클의 이미지**

칼럼 DBMS를 사용한 애플리케이션에 관해서

DBMS를 사용한 애플리케이션 개발 경험 없이 오라클 공부를 시작한 분들도 계실 겁니다. 그런 분들은 대부분 다음과 같은 의문이 드실 것입니다.

"시스템에서 데이터베이스는 어떤 식으로 사용되고 있는 걸까?"

"연구 논문이나 책들을 읽어 보면 오라클은 SQL문을 수행하기 위해 SQL*Plus라고 하는 것을 사용하는 듯한데, 이를 애플리케이션이 사용해도 괜찮은 걸까?"

우선, 첫 질문에 대한 대답은 '데이터베이스는 데이터를 보관하는 장소로 사용되고 있으며, 애플리케이션이 데이터를 보관하고(또는 변경하거나 꺼내고) 싶을 때 사용한다'입니다.

이미 알고 계시는 바와 같이 애플리케이션은 업무 처리나 화면 처리를 수행합니다. 애플리케이션은 이러한 작업들을 처리하다가 '이 데이터가 필요하다'라고 생각한 시점이 되면 데이터베이스에 접근해서 데이터를 꺼내 옵니다. 애플리케이션도 변수의 형태로 데이터를 보관할 수는 있지만, 이는 어디까지나 메모리에 임시로 저장하고 사용하는 것입니다. 그래서 일반적으로 애플리케이션이나 프로세스 간에 데이터를 공유할 수 없습니다. 따라서 종류가 많고 크기가 큰 데이터를 취급하는 시스템에서는 데이터베이스를 만들어 데이터 관리를 맡깁니다.

이어서 두 번째 질문의 대답은 '실제 시스템에서 애플리케이션이 SQL*Plus를 직접 사용하는 일은 거의 없다'입니다.

실제로 대부분의 시스템에서 애플리케이션은 JDBC(Java DataBase Connectivity)나 ODP.NET(Oracle Data Provider for .NET), Pro*C(오라클에서 사용하는 C언어 프리 컴파일러) 같은 것을 통해 오라클에서 SQL문을 수행합니다. SQL*Plus는 대부분 데이터베이스의 관리(테이블이나 인덱스의 생성 및 사람이 수동으로 직접 데이터를 검색하는 작업 등)를 위해서 사용합니다[주1]. 좀 더 이미지를 확실하게 잡을 수 있도록 애플리케이션에서 어떻게 JDBC를 사용하는지에 대한 간단한 예를 리스트 2.1에서 설명하겠습니다.

주1 단, 예외 사항으로 스크립트 안에 포함해 배치 처리 등에 사용하는 때도 있습니다.

리스트 2.1 자바에서 데이터베이스 사용하기

```
⋮
업무 처리 코드
⋮
⋮
여기서 데이터베이스의 데이터가 필요해짐
⋮
                                                    여기서 DBMS에 연결함
Connection conn = DriverManager.getConnection(url,user,pwd);
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT no, name FROM test WHERE ...");
                                                    이 부분에 의해 SQL문이 DBMS에서 전달됨.
                                                    DBMS는 SQL문을 받으면 처리해서 결과를
                                                    애플리케이션에 반환하며, 그 결과는 왼쪽의
                                                    ResultSet에 보관됨. 이후 애플리케이션은
                                                    이 ResultSet의 데이터를 사용해서 업무 처리나
                                                    화면 처리를 수행함
⋮
가져온 데이터를 이용해서 업무 처리 재개
⋮
⋮
업무 처리가 계속됨
⋮
```

2.2 데이터베이스의 데이터는 모두의 것

2.2.1 실제 모습에 가까운 오라클의 이미지

여기까지의 설명을 토대로 지금까지 배운 디스크 등의 이미지를 머릿속에 떠올리는 것이 가능해지셨는지요? 그러면 이제 그림 2.1을 좀 더 실제에 가까운 이미지로 만들어 보겠습니다(그림 2.2).

오라클에게 각종 프로그램(SQL*Plus를 포함)은 고객이라고 볼 수 있으며, 오라클은 SQL문과 그 결과의 데이터를 프로그램과 송수신합니다. DBMS인 오라클은 해당 데이

터를 디스크에 저장하고 관리합니다. 고객인 각종 프로그램은 여러 개 존재할 수 있으며, 의뢰를 받는 입장인 오라클의 프로세스도 여러 개 존재합니다. 이 '여러 개'라는 개념은 DBMS를 이해하는 데 있어서 매우 중요합니다.

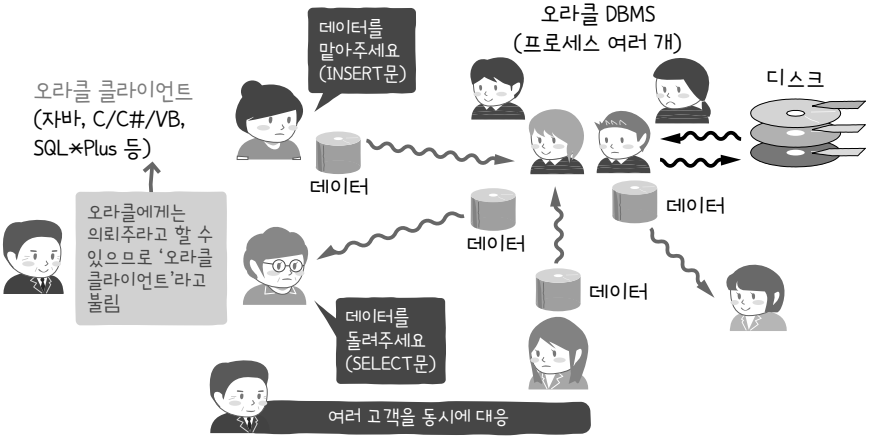

그림 2.2 **그림 2.1을 실제 모습에 가깝게 만든 오라클의 이미지**

데이터베이스를 사용하지 않는 애플리케이션의 프로그래밍에서는 각각의 프로세스가 자신이 가진 변수(데이터)를 처리하는 것이 일반적입니다. 같은 프로그램이 여러 개 실행되었다 하더라도 실제로 변수는 개개의 프로그램마다 존재하기 때문에 프로세스 간의 간섭을 신경을 쓸 필요가 없습니다(그림 2.3). 따라서 다른 프로그램을 고려하지 않고 데이터를 원하는 대로 건드려도 문제가 없습니다. 여러분도 프로그래밍할 때는 변수에 Lock을 건다든가, '다른 사람이 사용하고 있어서 이 변수는 사용하지 못할 수도 있어'와 같은 생각을 해본 적은 없었을 것입니다(단, 멀티스레드 프로그램은 제외합니다).

하지만 데이터베이스에서는 여러 프로세스나 사용자가 하나의 데이터베이스(데이터 집합)에 접근하기 때문에 다른 사용자나 애플리케이션을 신경 쓰지 않을 수 없습니다(그림 2.4).

CHAPTER 3

캐시와 공유 메모리

2장에서 오라클은 SQL의 처리를 주로 담당하는 서버 프로세스와 서버 프로세스를 지원하는 스태프인 백그라운드 프로세스들로 구성되어 있다고 설명을 드렸습니다. 이번 3장에서는 SQL문을 고속으로 처리하는 오라클의 데이터 캐시 기능(버퍼 캐시)에 관해서 살펴보겠습니다. 또한, 오라클의 프로세스가 캐시를 공유하기 위해 사용하는 특수한 메모리 기능(공유 메모리)에 관해서도 알아보겠습니다.

3.1 어째서 캐시가 필요한 것인가?

이번 장의 테마를 설명하기에 앞서 필요한 지식을 다시 한번 살펴보겠습니다. 디스크는 액추에이터(톤암)를 움직여서 데이터를 읽고 씁니다(그림 3.1). 오라클은 디스크에게 읽고 쓰는 것을 의뢰합니다. 오라클은 여러 개의 프로세스로 구성되어 있고 SQL문을 동시에 처리할 수 있습니다.

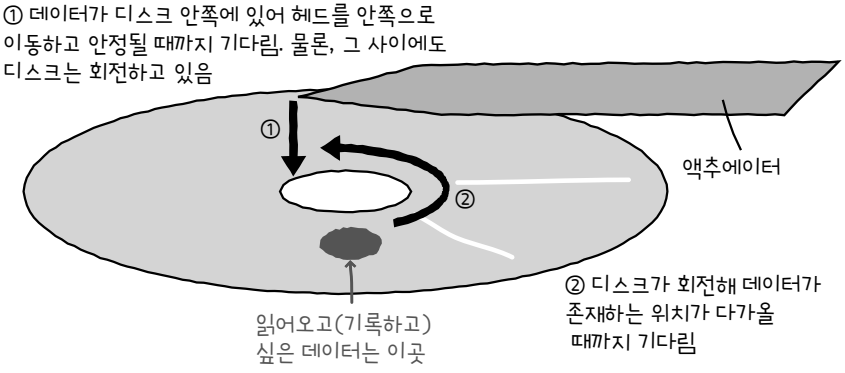

그림 3.1 **디스크의 I/O 처리에 필요한 동작**

또한 프로세스에는 각자의 역할이 있으며, SQL문을 빠르게 처리하기 위해 전념하는 서버 프로세스와 그것을 지원하는 백그라운드 프로세스가 존재합니다. 이 내용을 정리한 것이 그림 3.2입니다.

1장에서 디스크가 동작하는 것은 매우 느리고 I/O 1회에 10~20밀리초 정도 걸린다는 것을 설명했습니다. 그래서 오라클은 '캐시'라고 불리는 기술을 사용해 가능한 한 디스크에서 처리하지 않고, 메모리에서 처리하는 구조를 갖고 있습니다.

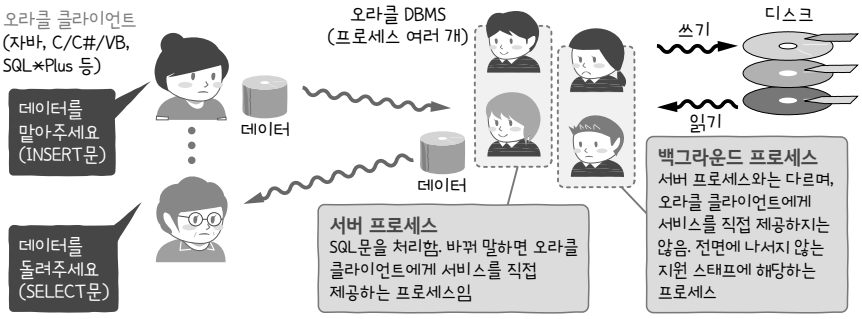

그림 3.2 **오라클의 구성(서버 프로세스와 백그라운드 프로세스)**

캐시는 가장 간단한 튜닝 항목이자 잘 알려진 기능 중 하나이지만, 아키텍처를 제대로 이해하고 있지 않으면 생각지도 못한 부분에서 '가져오고 싶었던 데이터가 캐시에 존재하지 않아 성능이 좋지 않다'와 같은 결과를 초래합니다. 캐시의 동작을 잘 이해해서 데이터베이스의 성능을 올려줍시다.

3.2 그래서 캐시란 대체 무엇인가?

먼저 일반적인 IT 용어로써 캐시가 무엇인지 알아보겠습니다.

일반적으로 캐시를 '작업장'이나 '작업대'에 비교하는 경우가 많습니다. 여러분이 일할 때 빈번하게 사용하는 도구나 책을 필요할 때마다 서랍이나 책장에서 꺼내어 사용하고, 끝나면 바로 다시 서랍이나 책장에 집어넣는 식으로 사용하는 분은 없을 것입니다. 빈번하게 사용한다면 책상 위나 손이 잘 닿는 곳에 놓아 두겠지요. 캐시의 목적도 이와 같습니다. 빈번하게 사용하는 데이터를 매번 디스크에서 꺼내오지 않고 캐시라고 불리는 메모리에 둠으로써 빠르게 사용할 수 있도록 하는 것입니다. 그림 3.3은 메모리(캐시), 디스크 그리고 CPU의 관계를 나타냈습니다.

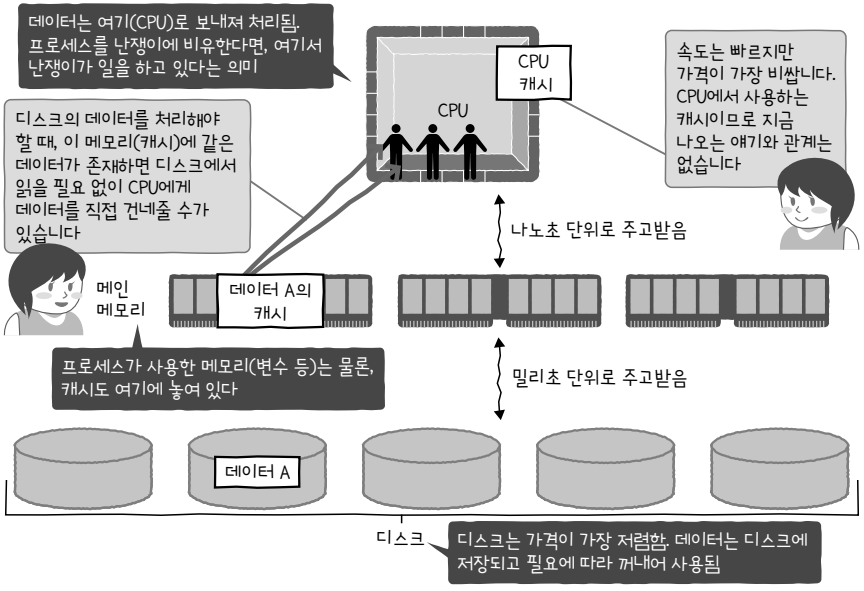

그림 3.3 **메모리(캐시), 디스크, 그리고 CPU의 관계**

다음으로 오라클에서의 데이터 캐시('버퍼 캐시'라고 불립니다) 동작을 살펴보겠습니다. 서버 프로세스가 원하는 데이터가 버퍼 캐시에 존재할 때(그림 3.4)와 존재하지 않을 때(그림 3.5)의 차이를 확인하세요. 캐시에 히트(hit)하지 않았을 때는 처리 속도가 느린 디스크에서 데이터를 읽어올 수밖에 없으므로 그만큼 SQL의 처리가 늦어집니다

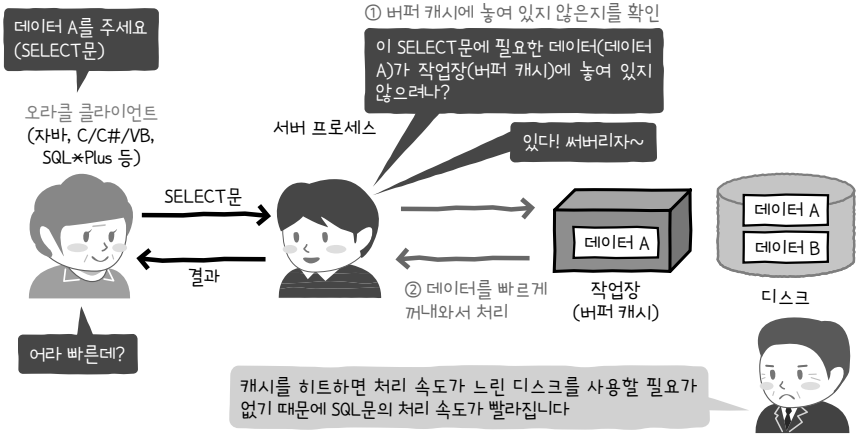

그림 3.4 **버퍼 캐시에 히트(데이터가 존재)할 때**

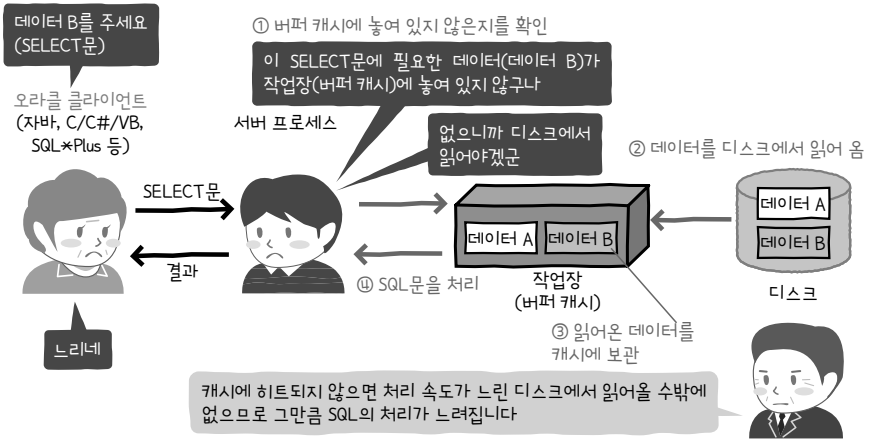

그림 3.5 **버퍼 캐시에 히트하지 않을(데이터가 존재하지 않을) 때**

3.3 데이터는 블록 단위로 관리

오라클은 '블록'이라고 하는 단위로 데이터를 관리합니다. I/O의 단위도 블록을 기반으로 하고 있으며, 버퍼 캐시도 블록 단위로 관리하고 있습니다. 블록은 단순하게 '정리용 상자'라고 생각하시면 됩니다. 일상생활에서도 크기가 작고 양이 많은 물건들을 정리할 때는 상자를 몇 개 준비하고 그 안에 보관하는 경우가 많습니다. 오라클의 데이터는 수 바이트에서 수천 바이트 이상의 여러 '로우(행)'로 존재하므로, 오라클 역시 블록이라고 하는 정리용 상자를 준비해서 보관하는 것입니다[주1]. 오라클 블록의 대략적인 구조는 그림 3.6과 같습니다.

주1 OS의 블록이 아니라 오라클의 독자적인 블록을 말합니다.

CHAPTER 5

오라클의 기동과 정지

4장에서 오라클의 SQL문에는 처리 방법이 기술되어 있지 않기 때문에 예측한 것을 기반으로 '실행 계획'이라는 데이터 처리 방법을 결정한다고 설명했습니다. 그 작업을 분석(파스)이라 부르고, 오라클에는 공유 풀이라 불리는 캐시가 있으며, 실행 계획을 캐시한다는 것도 설명했습니다. 이번 장에서는 이야기를 확 바꿔서 오라클의 기동과 정지에 관해 살펴보도록 하겠습니다. 오라클의 기동에는 네 가지 상태가 있다는 것과, 기동 처리의 흐름, 기동할 때 사용되는 파일 등에 대해 알아볼 것입니다.

5.1 기동과 정지를 왜 배워야 하는가?

OS와 마찬가지로 기동과 정지를 배우는 것은 오라클의 내부 구조를 이해하는 데 큰 도움이 됩니다. 기동할 때 어떤 파일을 어떻게 사용하는지, 의존 관계가 어떻게 되어 있는지를 안다면 내부 동작을 더 쉽게 이해할 수 있습니다. 더욱이 각종 파일이 손상된 장애에 대응해야 할 때도 내부 동작에 관한 지식을 알고 있다면 도움이 됩니다.

또한, 이번 장에서 설명하는 내용 일부는 윈도우(Windows)에서 동작하는 오라클에는 해당하지 않을 수도 있습니다. 단, 내부 구조는 대부분 같으며, 유닉스 쪽이 프로세스들을 살펴보기에 좀 더 쉬우므로 유닉스상의 오라클을 이해하는 것이 윈도우상의 오라클을 이해하는 지름길이라고 생각합니다. 따라서 윈도우 환경에서 오라클을 사용하시는 분들도 그냥 지나치지 마시고 꼭 읽어 보세요.

5.2 오라클의 기동/정지의 개요

이번 장에서도 이전 장들과 마찬가지로 오라클을 창고 회사라고 비유하겠습니다. 오라클의 기동은 '창고 회사의 업무 시작'이고, 오라클의 정지는 '창고 회사의 업무가 종료'하는 것에 해당합니다. 업무 시작과 종료는 큰 흐름으로 보면 다음과 같습니다.

창고 회사 오라클의 업무 시작 흐름

① 사원(영업 제외)이 출근함
② 창고에 관한 정보(관리 대장)를 조사함
③ 창고를 대충 훑어본 뒤, 문제가 없다면 창고를 열어 업무를 시작함

창고 회사 오라클의 업무 종료 흐름

① 업무(SQL이나 트랜잭션 등)가 끝나는 것을 기다린다. 단, 급할 때는 업무 중이더라도 업무 처리를 중지함[주1]

② 작업장의 물건(캐시상의 데이터)을 창고(파일)에 정리한다. 단, 급할 때는 작업장의 물건들을 정리하지 않음[주1]

③ 사원이 퇴근함(프로세스의 종료)

정리하면 그림 5.1과 같습니다.

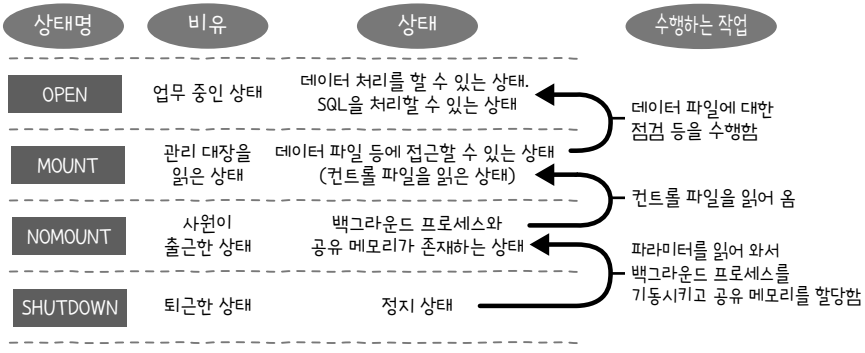

그림 5.1 **오라클의 기동에는 네 가지 상태가 있음**

5.3 업무의 시작에 해당하는 오라클의 기동

사원이 출근하지 않으면 어떤 업무도 시작할 수 없습니다. 따라서 업무를 시작하기 위해서는 사원이 먼저 출근해서 업무를 시작하기 위한 작업들을 해야 합니다. 창고 회

주1 종료하는 명령어의 옵션에서 '업무 중이더라도 업무 처리를 중지한다'와 '데이터를 정리하지 않는다'는 부분을 지정할 수 있습니다.

사 오라클에서는 관리 대장에 어떤 물건이 창고 어디에 있는지를 기록하고 있습니다. 이 관리 대장을 토대로 창고를 대충 훑어봤을 때 문제가 없다면 업무를 시작합니다. 그 후에 고객에게 요청이 오기 시작하면 영업 사원이 출근하여 고객의 요청에 대응합니다. 창고 회사 오라클의 업무 흐름은 현실의 창고 회사와는 조금 다르게 흘러갑니다.

그러면 창고 회사의 업무 시작 흐름과 실제 오라클의 기동을 그림 5.1을 보면서 비교해보겠습니다. 예를 들었던 이야기에서 '사원의 출근'은 백그라운드 프로세스의 생성과 공유 메모리의 확보에 해당하며, '창고에 관한 정보(관리 대장)를 본다'는 컨트롤 파일을 보는 것을 의미합니다. 컨트롤 파일이란, 데이터베이스의 구성 정보가 적혀 있는 파일로서 데이터베이스의 파일 경로 등을 알 수 있습니다. 그리고 '창고를 열어서 업무를 시작'하는 것이 SQL을 처리할 수 있는 상태로 만드는 작업입니다. 데이터 파일이나 REDO 로그 파일을 열어서 오라클이 내부적으로 사용하는 정보와 비교하여 문제가 있는지를 확인합니다. 데이터 파일은 이름대로 데이터가 보관된 파일을 의미하며, REDO 로그 파일은 데이터의 변경 이력을 보관하는 파일을 의미합니다.

5.4 인스턴스, 데이터베이스, 그리고 주요 파일의 구성

오라클을 관리하기 위한 단위로 인스턴스(instance)라는 용어를 사용하고 있습니다. 인스턴스라고 하면 객체지향 언어를 경험해보신 분들은 '엔터티(entity)'를 떠올리시겠지만, 오라클에서의 인스턴스는 백그라운드 프로세스+공유 메모리를 의미합니다. 그림 5.1의 'NOMOUNT' 상태는 인스턴스가 기동한 상태입니다. 인스턴스는 '데이터베이스를 관리하는 것(프로세스+메모리)'이며, 데이터베이스와는 다릅니다(그림 5.2).

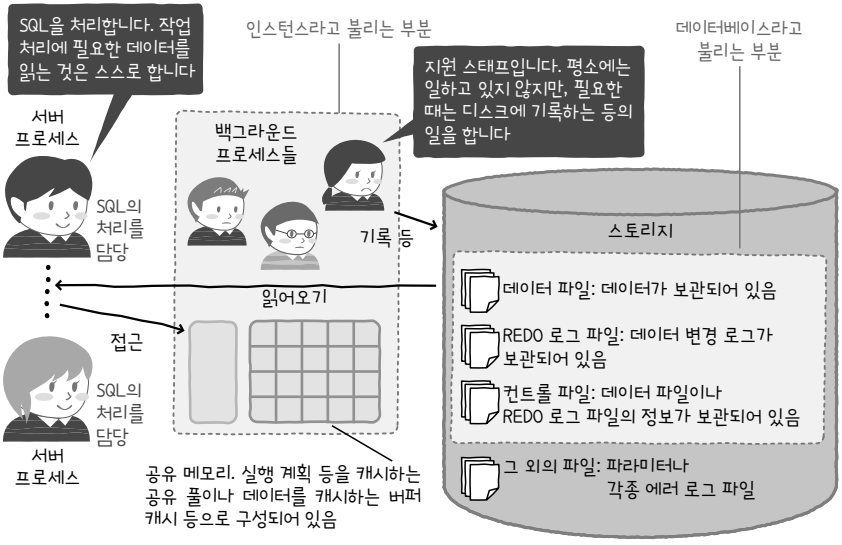

그림 5.2 데이터베이스란? 인스턴스란?

일반적으로 인스턴스와 데이터베이스는 일대일로 대응하므로 인스턴스를 데이터베이스라고 불러도 그렇게 큰 문제는 없습니다. 인스턴스가 기동되고 데이터베이스가 오픈된 상태를 '데이터베이스가 기동했다'라고 말합니다. 단, RAC(Real Application Clusters)[주2]를 사용하는 경우에는 인스턴스와 데이터베이스가 일대일로 대응하지 않기 때문에 인스턴스와 데이터베이스의 차이를 명확히 이해할 필요가 있습니다(그림 5.3).

주2 오라클 데이터베이스의 클러스터링 기능

CHAPTER 6

커넥션과 서버 프로세스의 생성

5장에서 오라클을 창고 회사에 비유해가며 데이터베이스가 어떻게 만들어지는지를 설명했지만, 이것만으로는 아직 SQL의 처리를 시작할 수 없습니다. SQL을 처리하기 위해서는 애플리케이션에서 오라클로의 커넥션을 맺고 서버 프로세스를 생성하는 것이 필요합니다. 그래서 이번 장에서는 애플리케이션에서 오라클로의 커넥션을 맺는 것에 관한 전반적인 내용과 서버 프로세스의 생성에 관해 설명하겠습니다.

6.1 애플리케이션에서의 접속을 왜 배워야 하는가?

그 이유는 애플리케이션에서 접속하는 것을 최적화하는 것만으로도 데이터베이스의 성능을 더욱 끌어올릴 수 있고, 애플리케이션에서 피해야만 하는 코딩 방식을 이해하는 데도 도움이 되기 때문입니다.

오라클은 애플리케이션 서버를 사용한 시스템이나 클라이언트/서버 형태[주1]의 시스템에서도 많이 사용되고 있습니다. 즉 오라클과 '오라클을 사용하는 애플리케이션'이 같은 서버 위에 있는 경우는 드물며, 애플리케이션(오라클 클라이언트)과 오라클이 네트워크를 통해서 통신하는 경우가 많습니다. 그러므로 접속 설정으로 인한 장애가 쉽게 발생하지만, 간단한 장애일 때는 아키텍처를 이해하고 있다면 대부분 쉽게 해결할 수 있습니다.

이번 장에서는 유닉스 환경을 기준으로 살펴보겠습니다. 따라서 설명하는 내용 중 일부는 윈도우에서 운영하는 오라클에는 해당되지 않을 수 있습니다. 단 윈도우용 오라클과 유닉스용 오라클의 내부 구조는 대부분 동일하지만, 유닉스 쪽 프로세스의 동작이 살펴보기에 좀 더 편합니다. 그러므로 유닉스 환경을 먼저 살펴보는 것이 윈도우 환경의 오라클을 이해하기 위한 지름길이라고 생각합니다. 따라서 윈도우 환경에서 오라클을 사용하는 분들도 꼭 읽어 주세요.

주1 클라이언트/서버(CS, Client/Server): 시스템 사용자가 사용하는 클라이언트 장비(주로 PC)에서 작동하는 클라이언트 프로그램을 '클라이언트'라고 부르며, 클라이언트에게 서비스를 제공하는 서버 장비에서 작동하는 서버 프로그램을 '서버'라고 부릅니다. 이 둘이 합쳐져서 클라이언트/서버를 구성합니다.

6.2 오라클의 접속 동작

6.2.1 소켓의 동작 이미지

오라클은 TCP/IP의 소켓(socket)을 네트워크 통신 수단으로 사용하고 있습니다. 그래서 소켓이 어떻게 동작하는지를 머릿속에 이미지로 떠올릴 수 있어야 합니다.

소켓을 사용하면 마치 전화처럼 다른 장비에 있는 프로그램과 통신할 수 있습니다. 장비 안에는 프로그램(프로세스)이 작동하고 있고, 그 프로그램이 수화기에 해당하는 소켓을 가지고 있는 모습을 머릿속에서 그려주시면 됩니다(그림 6.1).

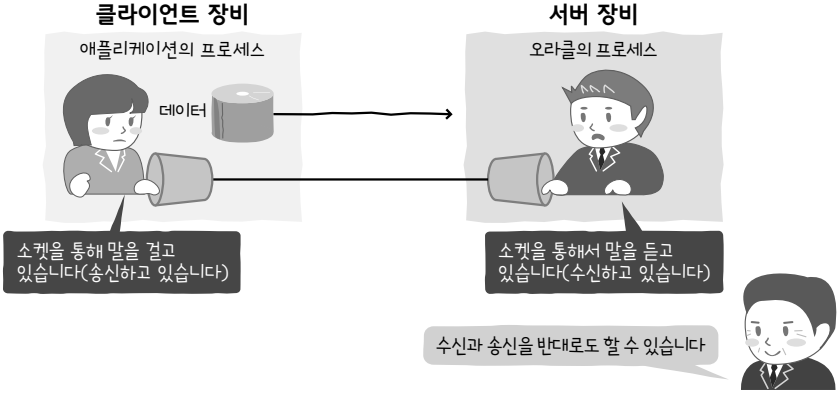

그림 6.1 **소켓의 동작 이미지**

한번 소켓을 만들어 두면 소켓을 읽고 쓰기만 해도 송수신을 구현할 수 있으므로 프로세스의 측면에서 보면 편리한 기능이라고 할 수 있습니다. 네트워크의 드라이버와 OS의 라이브러리가 송수신을 수행하며, 네트워크 안에는 여러 개의 소켓이 존재합니다. 그러면 어떻게 해야 원하는 대상에 틀리지 않고 연결할 수 있을까요? 소켓은 '주소(address)'와 '포트(port) 번호'라고 불리는 번호의 조합으로 식별할 수 있으며, 어떻게 연결하는지를 그림 6.2에서 동작과 함께 설명하겠습니다. 그림 6.2에서 중요한 부분은

연락이 오기만을 기다리고 있는 프로세스가 존재한다는 점과 연결할 때는 송신 측에서 '주소'와 '기다리고 있는 포트'를 반드시 지정해야 한다는 점입니다.

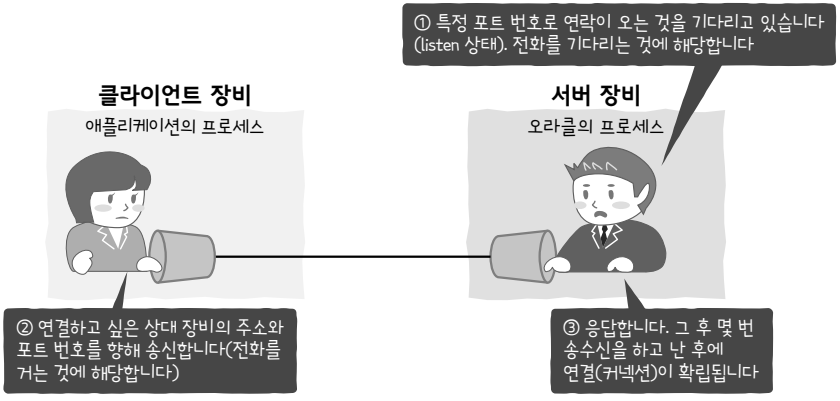

그림 6.2 **소켓이 연결되는 모습**

6.2.2 오라클에서 소켓의 동작

오라클에서도 소켓을 사용하고 있는 이상, 그림 6.2와 같이 동작합니다. 오라클에서는 수신을 기다리는 프로세스를 '리스너(listener)'라고 부릅니다(서버 프로세스가 아닙니다). 리스너로 접속하려는 프로세스는 업무 애플리케이션의 프로세스입니다. 오라클의 커넥션에 관해 간단히 정리한 것은 그림 6.3과 같습니다.

6.2.3 커넥션 처리 ①: 리스너 기동

커넥션 처리를 창고 회사 오라클에 비유하면서 상세히 설명하겠습니다. 리스너는 창고 회사 오라클의 접수 데스크입니다(그림 6.3의 ①에 해당). listener.ora 파일은 접수 데스크가 가지고 있는 '회사의 대표 번호' 및 '내선 전화번호부'이며, 하나의 리스너는 여러 개의 데이터베이스에 안내할 수 있습니다. 물론, 일반적인 상황에서는 하나의 리스너가 데이터베이스 한 개를 담당하게 합니다.

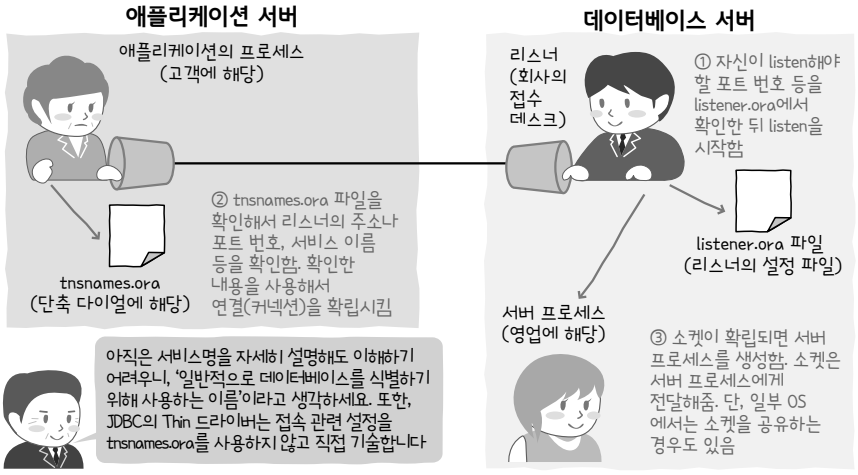

그림 6.3 **오라클 커넥션의 개요**

오라클은 기본적으로 리스너의 포트 번호로 1521을 사용하지만, 다른 애플리케이션과 충돌이 발생했을 때는 다른 번호를 사용해도 상관없습니다. 리스너의 설정[주2]이 끝나면, lsnrctl이라는 도구를 사용해서 리스너를 기동합니다(리스트 6.1). 리스너가 자신이 안내해야 하는 데이터베이스를 인식하는 방법으로는 listener.ora 파일에 기록되어 있는 설정을 읽거나, 데이터베이스가 자동으로 등록하는 방법이 있습니다. 일반적으로는 사용이 간단한 자동 등록을 선택해서 사용합니다. 이로써 접수 데스크 직원이 출근해서 업무를 시작할 준비가 되었습니다. 고객이 언제 전화를 걸어오더라도 업무를 수행할 수 있습니다.

주2 최근에는 주로 GUI 도구(NET Configuration Assistant, NETCA)를 사용해서 리스너나 오라클 클라이언트의 설정을 수행합니다. 이번에는 내부 동작을 소개하고 싶어서 설명하지 않았지만, 실제로 설정할 때는 도구를 사용해주세요.

APPENDIX A

유스케이스로 배우는 오라클

지금까지는 좀 더 많은 분들이 오라클의 기본적인 아키텍처를 이해할 수 있도록 가능한 한 오라클 용어를 사용하지 않고, 오라클의 명령어도 피하며 설명했습니다. 부록에서는 관점을 약간 바꾸어서 실제로 어떻게 사용하는지 명령어로 오라클을 조작해보며 좀 더 구체적으로 살펴보겠습니다. 지금까지 배워온 오라클의 아키텍처를 떠올리면서 실제로 어떤 명령어로 어떻게 조작하는지를 확인하세요.

A.1 A 씨에게 준비된 과제

A 씨는 직무 교육에서 선임에게 '데이터 파일 추가'를 혼자서 수행해보라는 과제를 받았습니다. A 씨가 어떤 방식으로 수행하는지 실제 명령어와 함께 살펴보겠습니다.

또한, A 씨가 실제로 만져볼 장비의 환경은 표 A.1과 같습니다. 기본적인 명령어는 버전이 다르더라도 크게 바뀌지는 않지만, 실행할 때는 대상 버전의 매뉴얼을 확인하세요.

표 A.1 환경 정보

항목	값
OS	Oracle Linux Server release 6.8
오라클 데이터베이스 버전	18c
ORACLE_HOME	/u01/app/oracle/product/18.0.0/dbhome_1
ORACLE_SID※	ORCL

※ 인스턴스명. 싱글 구성에서는 데이터베이스명이 ORACLE_SID다.

A.2 오라클의 기동

⇨ 참고 5장 오라클의 기동과 정지

지금은 사용하지 않는 검증용 데이터베이스를 빌렸으니까 거기서 테스트해보자. 일단은 기동부터 시작해야겠네.

오라클을 조작하기 위한 도구인 SQL*Plus를 사용해서 오라클에 접속합니다. 접속할 때는 SQL*Plus가 접속할 데이터베이스 인스턴스 및 SQL*Plus의 실행 파일이 있는 경로를 인식하기 위해서 리스트 A.1과 같이 환경 변수를 설정해야 합니다.

리스트 A.1 환경 변수의 설정과 오라클 접속

```
## 환경 변수 설정
[오라클 계정@OS 프롬프트]$ export ORACLE_SID=ORCL
[오라클 계정@OS 프롬프트]$ export ORACLE_HOME=/u01/app/oracle/product/↵
18.0.0/dbhome_1
[오라클 계정@OS 프롬프트]$ export PATH=$ORACLE_HOME/bin:$PATH

## SQL*Plus에 접속
[오라클 계정@OS 프롬프트]$ sqlplus / as sysdba
SQL*Plus: Release 18.0.0.0.0 - Production on Thu Dec 27 18:06:08 2018
Version 18.3.0.0.0

Copyright (c) 1982, 2018, Oracle.  All rights reserved.


Connected to:
Oracle Database 18c Enterprise Edition Release 18.0.0.0.0 - Production
Version 18.3.0.0.0
Connect to an idle instance.
SQL>
```

메모

이후에 나오는 내용은 환경 변수를 이미 설정했다는 전제하에 명령어를 수행합니다. 만약 서버에 다시 로그인했을 때는 잊지 말고 환경 변수를 설정하세요.

SQL*Plus에 연결됐네. 그럼 오라클을 기동해보자. 혹시 모르니 잘 기동되었는지 확인도 해 두자.

그러면 오라클을 기동하고(리스트 A.2), 데이터를 처리할 수 있는 OPEN 상태인 것을 확인해보겠습니다.

리스트 A.2 오라클의 기동

```
SQL> startup
ORACLE instance started.

Total System Global Area 2768240008 bytes
Fixed Size                  8932744 bytes
Variable Size             704643072 bytes
Database Buffers         1979711488 bytes
Redo Buffers               74952704 bytes
Database mounted.
Database opened.

-- 기동 확인
SQL> select instance_name, status from v$instance;
INSTANCE_NAME    STATUS
---------------- ------------
ORCL             OPEN
```

A.3 리스너를 통한 접속

⇨ 참고 6장 커넥션과 서버 프로세스의 생성

아! 기왕 한 김에 애플리케이션에서 접속하는 걸로 가정하고 리스너를 통해서 접속해봐야겠다.

우선은 리스너를 기동(리스트 A.3)하고 애플리케이션에서 접속할 수 있는 상태로 만듭니다.

리스트 A.3 리스너의 기동

```
[오라클 계정@OS 프롬프트]$ lsnrctl start listener
```

기본 리스너를 기동할 때는 리스너명을 생략할 수 있음

메모

6장에서는 리스너명을 생략한 명령어를 소개했습니다. 리스너명을 생략하면 기본 리스너인 'listener'가 기동됩니다.

인스턴스를 기동할 때 리스너가 기동되지는 않으며, 자동 등록 방식(동적 서비스 등록, dynamic registration)을 사용하고 있을 때 리스너에게 자신이 안내해야 할 데이터베이스를 명시적으로 알려주기 위해 리스트 A.4의 명령어를 실행합니다.

메모

실제로는 이 명령어를 실행하지 않더라도 정기적으로 체크하기 때문에 조금만 기다리면 자동으로 등록됩니다.

리스트 A.4 서비스 등록

```
[오라클 계정@OS 프롬프트]$ sqlplus / as sysdba

SQL> alter system register;
```

리스너는 기동되었으니 접속할 데이터베이스의 정보를 확인하고 접속해보자.

호스트명(또는 IP 주소), 리스닝 포트, 서비스명을 확인하고 접속합니다(리스트 A.5). 접속 후에 인스턴스명을 확인하여 원하던 데이터베이스에 잘 접속했는지를 확인합니다.

리스너의 상태를 확인했을 때 접속하고 싶은 인스턴스의 서비스명이 표시되지 않으면 리스너는 자신이 안내해야 할 데이터베이스를 인식할 수 없습니다. 그러므로 앞에서 설명했던 'alter system register'를 실행해보거나, listener.ora 파일 등의 설정을 다시 한번 살펴봐야 합니다.